NATIONAL INSTITUTE of STANDARDS and TECHNOLOGY/
NATIONAL COMPUTER SECURITY CENTER

C.I+P NIST/NCSC c.1
v.1

13th National Computer Security Conference

Omni Shoreham Hotel
Washington, D.C.
1 - 4 October, 1990

Proceedings

VOLUME I

Information Systems Security:

Standards - The Key to the Future

20090327405

Welcome!

The National Computer Security Center (NCSC) and the National Computer Systems Laboratory (NCSL) are pleased to welcome you to the Thirteenth Annual National Computer Security Conference. We believe that the Conference will stimulate a vital and dynamic exchange of information and foster an understanding of emerging technologies.

The theme for this year's conference, "Information Systems Security: Standards -- The Key to the Future," reflects the continuing importance of the broader information systems security issues facing us. At the heart of these issues are two items which will receive special emphasis this week -- Information Systems Security Criteria (and how it affects us) and Education, Training, and Awareness. We are working together, in the Government, Industry, and Academe, in cooperative efforts to improve and expand the state-of-the-art technology to information systems security. This year we are pleased to present a new track by the information security educators. These presentations will provide you with some cost-effective as well as innovative ideas in developing your own on-site information-systems-security education programs. Additionally, we will be presenting an educational program which addresses the automated information security responsibilities. This educational program will refresh us with the perspectives of the past, and will project directions of the future.

We firmly believe that security awareness and responsibility are the cornerstone of any information security program. For our collective success, we ask that you reflect on the ideas and information presented this week; then share this information with your peers, your management, your administration, and your customers. By sharing this information, we will develop a stronger knowledge base for tomorrow's foundations.

[illegible]
JAMES H. BURROWS
Director
National Computer Systems Laboratory

Patrick R. Gallagher
PATRICK R. GALLAGHER, JR
Director
National Computer Security Center

Conference Referees

Dr. Marshall Abrams	*The MITRE Corporation*
James P. Anderson	*James P. Anderson Company*
Jon Arneson	*National Institute of Standards & Technology*
Devolyn Arnold	*National Computer Security Center*
James Arnold	*National Computer Security Center*
Al Arsenault	*National Computer Security Center*
Victoria Ashby	*The MITRE Corporation*
Elaine Barker	*National Institute of Standards & Technology*
Dr. D. Elliott Bell	*Trusted Information Systems, Inc.*
Greg Bergren	*National Computer Security Center*
James Birch	*Secure Systems, Incorporated*
Earl Boebert	*Secure Computing Technology Corporation*
Dr. Dennis Branstad	*National Institute of Standards & Technology*
Dr. John Campbell	*National Computer Security Center*
Dr. Steve Crocker	*Trusted Information Systems, Inc.*
Dr. Dorothy Denning	*Digital Equipment Corporation*
Donna Dodson	*National Institute of Standards & Technology*
Greg Elkmann	*National Security Agency*
Ellen Flahaven	*National Institute of Standards & Technology*
Dan Gambel	*Grumann Data Systems*
Dain Gary	*Mellon National Bank*
Bill Geer	*National Computer Security Center*
Virgil Gibson	*Grumann Data Systems*
Dennis Gilbert	*National Institute of Standards and Technology*
Irene Gilbert	*National Institute of Standards and Technology*
Dr. Virgil Gligor	*University of Maryland*
Capt James Goldston, USAF	*National Computer Security Center*
Dr. Joshua Guttman	*The MITRE Corporation*
Dr. Grace Hammonds	*AGCS, Inc.*
Douglas Hardie	*Unisys Corporation*
Ronda Henning	*Harris Corporation*
Jack Holleran	*National Computer Security Center*
Jim Houser	*National Computer Security Center*
Russ Housley	*XEROX*
Dr. Dale Johnson	*The MITRE Corporation*
Carole Jordan	*Defense Investigative Service*
Sharon Keller	*National Institute of Standards & Technology*
Leslee LaFountain	*National Computer Security Center*
Steve LaFountain	*National Computer Security Center*
Paul Lambert	*Motorola GEG*
Carl Landwehr	*Naval Research Laboratory*
Robert Lau	*National Computer Security Center*

Dr. Theodore Lee	*Trusted Information Systems, Inc.*
Nina Lewis	*University of California, Santa Barbara*
Steve Lipner	*Digital Equipment Corporation*
Terry Losonsky	*National Security Agency*
Dr. Vic Maconachy	*National Security Agency*
Barbara Mayer	*Trusted Information Systems, Inc.*
Frank Mayer	*Sparta*
Vin McLellan	*Boston University*
Catherine A. Meadows	*Naval Research Laboratory*
Dr. Jonathan Millen	*The MITRE Corporation*
William H Murray	*Independent Consultant*
Eugene Myers	*National Computer Security Center*
Ruth Nelson	*GTE*
Dr. Peter Neumann	*SRI International*
Steven Padilla	*Trusted Information Systems, Inc.*
Nick Pantiuk	*Grumann Data Systems*
Donn Parker	*SRI International*
Rich Petthia	*Software Engineering Institute*
Dr. Charles Pfleeger	*Trusted Information Systems, Inc.*
Jerrold Powell	*Department of the Treasury*
Maria Pozzo	*University of California, Los Angeles*
Michael Rinick	*Central Intelligence Agency*
Ken Rowe	*National Computer Security Center*
Prof Ravi Sandhu	*George Mason University*
Marv Schaefer	*Trusted Information Systems, Inc.*
Dr. Roger Schell	*GEMINI*
Dan Schnackenberg	*Boeing Aerospace*
Miles Smid	*National Institute of Standards & Technology*
Suzanne Smith	*Los Alamos National Laboratory*
Brian Snow	*National Security Agency*
Prof. Gene Spafford	*Purdue University*
Dr. Dennis Steinauer	*National Institute of Standards & Technology*
Freddie Stewart	*ANSER*
Dr. Cliff Stoll	*Harvard - Smithsonian Center for Astrophysics*
Marianne Swanson	*National Institute of Standards & Technology*
Mario Tinto	*National Computer Security Center*
Ann Todd	*National Institute of Standards & Technology*
Eugene Troy	*National Institute of Standards & Technology*
LTC Ray Vaughn, USA	*National Computer Security Center*
Grant Wagner	*National Computer Security Center*
Jill Walsh	*INCO, Inc.*
Wayne Weingaertner	*National Computer Security Center*
Roger Westman	*INCO, Inc.*
Roy Wood	*National Computer Security Center*

Thirteenth National Computer Security Conference
October 1-4, 1990
Washington, DC

Table of Contents

TRACK C-II - Management & Administration

Educator Sessions

Alternate Papers

Student Papers

UNIX SYSTEM V WITH B2 SECURITY

Craig Rubin

AT&T Bell Laboratories
190 River Road, Summit NJ 07901

Abstract

This paper describes the feature changes needed for UNIX® System V to meet the Trusted Computer Systems Evaluation Criteria (TCSEC) [1] B2-level requirements while still maintaining original UNIX System design objectives and flexibility. Implications for users and administrators are discussed.

1. Overview

Traditional UNIX System users contend that the introduction of B2-level security features will negate many positive aspects of the UNIX System; security purists doubt that the UNIX System can meet the B2 criteria [2]. This paper addresses these issues, discusses the B2 features that have been added to UNIX System V, and explains the effects of these features on users and administrators.

2. Background

The UNIX System was originally developed in an open R&D environment in which a paramount concern was the free and easy exchange of information. Unpassworded guest logins, unprotected source and system files, and unrestricted dial in lines are typical in such an environment. Although security features were available, they were usually viewed as unfriendly and consequently were rarely used.

Lax security administration was only made worse by operator errors, an inadequate amount of security and administrative documentation, software holes through which hackers could gain unauthorized privileges, and the ability of unprivileged users to read the password file (which contained encrypted versions of the passwords).

UNIX is a registered trademark of AT&T.

3. *Motivation*

Customer demand for improved operating system security motivated the development of improved security in UNIX System V. Security requirements specified by foreign and domestic governments, the business sector, and other security-conscious data processing environments provided the impetus for standards and policy groups (such as IEEE P1003.6, ISO, X/OPEN, and the NCSC TRUSIX working group) to address security needs as they apply to the UNIX System.

4. *Goals*

AT&T has committed to produce a UNIX System that meets the needs of both government and commercial data processing operations. The goal of this system is to provide all (TCSEC) B2-level features, close any known security holes, and include improved operational procedures and monitoring tools. These features will be incorporated into the standard UNIX System V product, preferably as options, allowing sites to determine the best mix for size and performance constraints. Another critical factor is compatibility with existing releases of UNIX System V.

In addition to full B2 functionality, the discretionary access control (DAC) and trusted facility management (TFM) B3-level features will be available in the standard System V product.

5. *Approach*

AT&T's approach in addressing the security requirements has been to work closely with UNIX International to identify needs and evaluate functionality. A parallel effort has proceeded with government and industry leaders to establish standards through bodies such as IEEE POSIX and X/OPEN.

6. *Operating System Engineering Improvements*

Operating system engineering improvements go beyond individual feature development and involve changes in the structure and architecture of UNIX System V that result in improved maintainability, performance, flexibility, and portability. Typically, though not always, these improvements will be visible only to system porters and not to end users or application developers. Thus, while such improvements may benefit end users and developers, they are of direct interest to UNIX System V source code customers who plan to port or change the operating system.

The UNIX System has been renowned as a modular, highly portable operating system. To meet the exacting requirements on operating system modularity at the B2-level, however, the UNIX System V operating system will be further partitioned into modules.

Improved modularity impacts more than the security feature. It improves the entire operating system and benefits all source code customers. Modular code is easier to interpret, maintain, and port.

A modular system is one that is internally structured into well-defined, independent modules, where each module [3]:

— has a well defined function,

— has a well defined interface,

— has well defined parameters, and

— is called whenever its function is required.

Other related modularity improvements include restricting the use of global variables and allowing the use of nested header files. A tool was created to assist in the detection and examination of all global variables in the kernel. The information generated by this tool allowed many global variables to be changed to a local scope and provided justification for those global variables that remained.

7. Feature Specific Requirements

The following work is required for the development of a B2-level system and will require procedural changes on the part of users and/or administrators.

7.1 System Architecture

The system architecture criteria places several requirements on the internal design and structure of the Trusted Computing Base (TCB). A key feature that will be introduced in this area is a least privilege mechanism that breaks up the single super-user privilege into many smaller, well-defined privileges. A second new architectural feature is the aforementioned improved system modularity. These changes will have little procedural impact on users and administrators, however they will improve system assurance.

7.2 Discretionary Access Control (DAC)

The existing UNIX System provides the ability to distinguish permissions for the object owner, object owning group, and all others. This mechanism may be viewed as a fixed length, three entry, Access Control List (ACL). In order to meet the B3-level requirements, the B2 system provides full access control lists. This new mechanism interacts compatibly with the existing mechanism, preserves the meaning of the existing file permission bits, and allows the existing mechanism to work as before [4].

7.3 Security Labels

All processes, files, and IPC objects must have a security label. Device types must be designated as single-level (such as a tty) or multilevel (such as a special device file for a disk partition). When exporting data to a multilevel device, the data's sensitivity label will be exported with the data. This is not necessary with a single-level device.

7.4 Mandatory Access Control (MAC)

In addition to the Discretionary Access Control (DAC) facility, a Mandatory Access Control (MAC) facility is required. While the DAC mechanism allows permissions to be set at the discretion of the owner of an object and enforced by the system, the MAC mechanism is set by the system administrator and enforced by the system. The existing UNIX System did not provide any mechanism for MAC. The mandatory access control policy follows a modified Bell-LaPadula model [5] that can be summarized as "read equal or down" and "write equal." For instance, a process at level "top-secret" can read a file at level "secret," and a process at level "secret" would only be able to write to a file at level "secret."

Administrators are responsible for determining and setting up the discrete set of labels at which a user can log in. An administrator also sets a login level range on a terminal line, such that when a user attempts to login, the label specified by the user must dominate the login-low label on the terminal line and in turn be dominated by the login-high label on the terminal line.

Since the addition of mandatory access control labels will limit creation of files in a directory to processes at the same level as the directory, a new type of directory referred to as a multilevel directory (MLD) has been added to the system. A multilevel directory involves the addition of an extra, normally hidden layer in the directory hierarchy for directories.

When a process attempts to reference an MLD (e.g., /tmp) the kernel automatically translates this reference to a level-specific, hidden subdirectory known as the effective directory. For ease of use the effective directory is created automatically by the kernel if it does not already exist. An effective directory will exist for each process level which has accessed the multilevel directory. Since the effective directory is hidden, the process can not directly access it. However, some processes will have to perform maintenance on multilevel directories so they must be able to determine which effective directories are present and be able to directly access these directories. This is known as the real view of the multilevel directory and is accomplished by the process placing itself in real multilevel directory mode. The only difference from the existing method is that the process can not see all files in the MLD directory, but only files at the same label as the process. The standard MAC and DAC checks apply to multilevel directories and the files that they contain. This implementation conforms to the MAC policy, in that a process should only be able to see files (such as in /tmp) that are dominated by the label of the process. Public directories (writable and readable by all processes), such as /tmp must be MLDs. The use of MLDs eliminates many covert channels associated with public directories.

The mandatory access control facility is used along with the discretionary access control facility to mediate access to objects. When an access is attempted, both mandatory access and discretionary access checks are performed. If both checks pass, access is then granted.

7.5 Identification and Authentication

The existing Identification and Authentication mechanism (login and password) meets most of the B2-level requirements. However, the method had to be modified to support the new features being introduced. These include the specification of a MAC label at login time and recording login attempts in the audit trail. Furthermore, to support a trusted path, users are able to change their password only at login time, as this is the only time that the user will have a trusted path.

7.6 Audit

The existing UNIX System's accounting mechanism does not produce the finely-grained information that is required by the B2 criteria. Therefore, a new auditing mechanism was added.

The audit mechanism will have no impact on users. Administrators will select and set the events that are to be audited for all users and optionally set an audit mask for specific users. The events audited for any specific user can be changed by the logged-in administrator in real time. The system provides facilities for both pre-selection and post-selection of audit event data.

7.7 Object Reuse

When a storage object is assigned to a subject, the object must contain no data. This requirement is met by the existing UNIX System V.

7.8 Trusted Path

A trusted communication path between the TCB and a subject is required. This affects both the user and administrator. The administrator is responsible for defining a secure attention key (sak) for each terminal line. When a user or administrator wants to log in to the machine, they must first enter the sak. When the system detects the sak, it will initiate the login sequence on the terminal. If login is not completed within the login timeout period, the login program will terminate and the user is once again required to enter the sak in order to reinitiate the login process.

7.9 System Integrity

Proper operation of the hardware and firmware parts of a system must be verifiable. This will be achieved with the existing diagnostics available with the evaluated machine.

7.10 Trusted Facility Management

Separate operator and administrator functions are required at B2; to meet B3 requirements, a security administrator function must also be added. The current capabilities of the super-user login were separated into the aforementioned functions through a database maintained by the trusted system programmer. This Trusted Facility Management (TFM) database contains information specifying the commands that may be executed with privilege by various administrators. This database must be properly configured by the trusted system programmer before the system is used in the B2 configuration. A command that mediates the access given to a particular program must be used by the administrator to perform privileged operations.

8. Non-Feature Specific Requirements

The following work is required for the development of the B2-level system; this will not require any direct action on the part of users or administrators.

8.1 Covert Channel Analysis

A thorough search must be performed to identify all covert storage channels and determine their bandwidths. Covert channels must be closed, reduced, audited, or documented depending on the bandwidth. For those being audited, the auditor must be aware of the potential disclosure that may occur through the use of these covert channels and watch for their use in the audit trail.

8.2 Design Specification and Verification

A formal model of the security policy enforced by the TCB is required. This model was developed by AT&T with the NCSC TRUSIX working group. Also, a complete specification that describes the TCB "in terms of exceptions, error messages, and effects" is required for a B2 system. The model and the specification will be shown to be consistent.

8.3 Configuration Management

A configuration management system for use during the development and maintenance of the TCB is required. All documentation, code, and hardware must be controlled by this system. Tools to generate a new version of a system and to compare versions must be available. These requirements will be achieved by several complementary methods described in a product development methodology handbook. These methods (which are used for code, documents, and hardware) include the use of a source code control system, a change tracking system, and a change control committee.

8.4 Testing

Extensive testing of the security features at each level is required. In general, the testing must:

1. show that security features work as documented,
2. show that there are not obvious ways to bypass security mechanisms, and
3. show that identified flaws have been removed and that no new ones have been introduced.

The system should also be compatible with the existing UNIX System and with current standards such as POSIX. The development organization runs multiple test suites on the system to test for conformance to all of the required objectives. To test the new features that are being introduced, new test suites were added or existing test suites modified.

8.5 Documentation

The documentation required to describe the security mechanism is incorporated into the existing UNIX System documentation. The following list roughly summarizes the end user documentation required at the B2-level and identifies the existing UNIX System documents that it appears in.

- The UNIX System V User's Guide, along with manual pages in the UNIX System V User's Reference Manual, contains the information required of a security features user's guide. This information explains how a user is affected by the security mechanisms and their proper use (e.g., MAC and DAC). In addition, changes required for the existing system to meet the B2-level and their impact on the user are described (e.g., changes to the line printer subsystem).

- The UNIX System V System Administrator's Guide, Programmer's Guide, Programmer's Reference Manual, System Administrator's Reference Manual, and the newly introduced Audit Trail Administrator's Guide contain the information required in a trusted facility manual.

8.6 New File System Type

A new file system type, the Secure File System (SFS) has been added as the means of supporting the MAC and DAC security capabilities described previously. The new file system type is based on the UNIX File System (UFS) that was introduced with UNIX System V Release 4. The features of the new file system type that were added specifically to support security are:

- increasing the size of the inode so that labels and ACL's can reside in the inode, and
- adding support for multilevel directories (e.g., /tmp).

This addition will be invisible to users, and will require minor changes for administrators. Since the existing UNIX System already supports various file system types, administrators are familiar with different file system types.

On a non-B2 system, the new file system type can be mounted read-only as an ordinary UFS file system. Similarly, an ordinary file system can be mounted on a secure system as a single-level file system, and will not support ACLs. This is primarily needed to support the transition to a B2 secure system.

9. Conversion to a B2 System

Conversion of an ordinary system to a B2 secure system will require administrative set-up, especially in the areas of MAC, TFM, and privilege.

10. Summary

Although numerous changes have been made to incorporate the B2-level security features into UNIX System V, the system will still maintain the original UNIX System design objectives and provide the flexibility expected by users and administrators.

11. REFERENCES

[1] Department of Defense. *Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD, December, 1985.

[2] Sibert, W. O., Traxler, H. M., Wagner, G. M., Downs, D. D., Elliot, K. B., and Glass, J. J.: *UNIX And B2: Are They Compatible?*, in *Proceedings Of The 10th National Computer Security Conference*, September 21-24 1987, pp. 142-149.

[3] Stevens, W., Meyers, G., and Constantine, L.: *Structured Design, IBM Systems Journal*, Vol 13, No. 2, May 1974, pp. 115-139.

[4] National Computer Security Center *Trusted UNIX Working Group (TRUSIX), Rationale For Selecting Access Control List Features For The UNIX System*, NCSC-TG-020-A, VERSION-1, August 18, 1989.

[5] Bell, D. E. and LaPadula, L. J. *Secure Computer System: Unified Exposition and Multics Interpretation*, MITRE Corporation, MTR-2997, March 1976.

COVERT STORAGE CHANNEL ANALYSIS: A WORKED EXAMPLE

Timothy E. Levin and Albert Tao
Gemini Computers, Inc.
Carmel, California

Steven J. Padilla [*]
Trusted Information Systems, Inc.
Glenwood, Maryland

Abstract This paper presents an overview of the methodology used in a formal covert storage channel analysis of the GEMSOS Security Kernel. A synthesis of several well known covert channel approaches has been applied: the resulting methodology provides a significant reduction in effort relative to the techniques from which it was derived.

The method involves reducing the analysis to the information flows that can produce covert channels. The analysis is shown to be effective for systems whose direct illegal flows (as opposed to transitive flows) are both limitable and auditable.

A similar informal analysis technique is briefly described. This informal analysis can be used independently from the formal analysis or in conjunction with the formal analysis for confirmation of results.

Background

The Gemini GEMSOS TCB is in evaluation, targeted at the Trusted Computer System Evaluation Criteria class A1 rating. As part of this evaluation, a class A1 Trusted Network Interpretation [TNI87] "M-Component" evaluation of a product based on the kernel portion of the TCB is taking place as an incremental step in the overall TCB evaluation. This product is the GEMSOS Trusted Network Processor (GTNP).

The GEMSOS Trusted Network Processor (GTNP) consists of the GEMSOS Security Kernel and hardware base [SCHEL85], along with a non-kernel interface to define and support trusted and single-level processes [THOM90]. The GEMSOS Kernel provides a mandatory access control reference monitor. For the class A1 evaluation, a covert storage channel analysis has been performed on the GEMSOS Kernel. This report summarizes the approach used in that effort and is offered as a worked example of an efficient means of doing covert storage channel analysis.

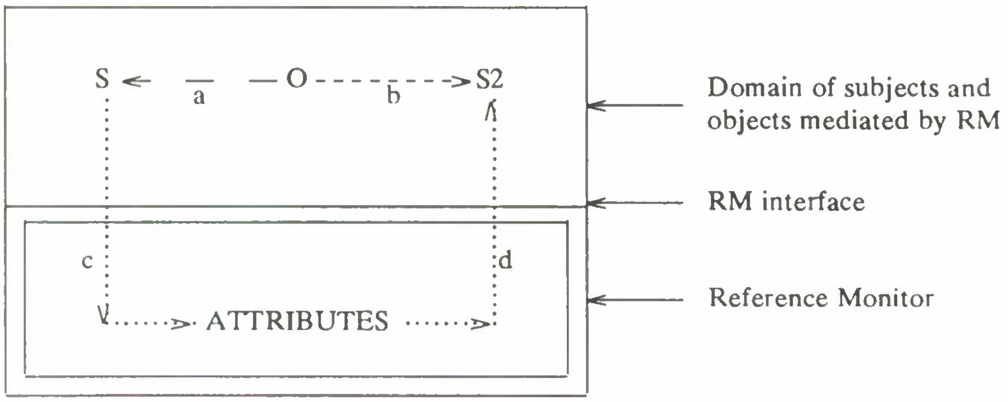
Covert Channel Analysis Within the Reference Monitor Paradigm

Analysis of information flow is examined in this paper relative to the concept of the reference monitor ("RM")[TCSEC]. Within this context we can identify a taxonomy of information flows. Flows can be classified as legal or illegal relative to the security policy. Some illegal flows are not exploitable at the RM interface; these are not of concern to this discussion. Exploitable illegal flows can be classified as either covert channels or RM flaws (discussed below).

The RM creates the subjects and data storage objects of the system, and mediates access between them. The RM maintains "attributes" of subjects, objects and system resources. These attributes are defined to

[*] This paper reflects work performed while Mr. Padilla was an employee of Gemini Computers, Inc.

be outside of the domain of subjects and objects protected by the RM. Operations that the reference monitor supports can be classified as legal (i.e., correct mediation of subjects' access to storage objects) or flawed (i.e. a subject bypasses the reference monitor, or there is improper mediation in a subject's access to a storage object, as shown for operation "b" in the following diagram). Illegal flows resulting from flawed operations are RM flaws. The next diagram illustrates these differences.



S = subject	Level of S = syshi
O = object	Level of O = syshi
S2 = subject	Level of S2 = syslo
- - = RM flaw	Level of ATTRIBUTE = syshi
... = covert channel	

a, b, c and d are RM operations with disjoint effects (the effects are flows represented by arrows going to or from the calling subject). a, c and d are legal operations (RM functions correctly). b is an operation with an RM flaw (access is mediated incorrectly). b and d are operations which produce illegal flows.

An example of a or b is a file-open operation which returns data from the object. An example of c is an operation to change a file's size. An example of d is an operation to return a file's size.

In this paradigm, covert channels result from information passing through a system attribute which is not mediated as a storage object. Examples of system attributes might be: file size, volume space availability, or CPU availability. A covert channel is induced and interpreted by a series of legal operations which reference such attributes.

Covert storage channels are distinct from covert timing channels. The manner in which the information from the covert channel leaves the reference monitor determines whether the channel is a storage or timing channel. For storage channels, information is passed out of the reference monitor through a change to a storage location (e.g., return value or error message); for timing channels, the information is returned outside of the reference monitor through a delay (i.e., a measurable change in response time).

Typically, timing channels and some storage channels are created through contention for finite system resources (the availability of the resource is a system attribute). In this type of channel, a high-level subject signals to a low-level subject by modulating its use of the resource, thus controlling the low-level subject's ability to use the resource. If contention is resolved through a delay to the low-level calling subject (e.g., the CPU is busy and the subject is made to wait), a timing channel is created. On the other hand, if the low-level subject receives a return value or error message when the resource is not available(e.g., "disk_full" error message), then we consider it a storage channel. One approach to closing resource exhaustion channels is to partition the resources by process or by security level (see "Channel Bandwidth Estimation" and "Informal Identification of Covert Storage Channels," below). This approach can have a significant negative effect on system performance when applied to timing channels.

Other storage channels are not based on resource contention. In these cases, system attributes other than "resource-busy" (for some given resource) are read and written. Information can thus be channeled through a change to file size, or object security classification. These are storage channels since the information passes out of the system through a change to a storage location (i.e., the change in file size is returned to the caller in a output parameter, or is signalled through the return of an error message). These channels can be (and in a secure system should be) avoided through rigorous system security engineering.

In a complete FTLS (as required by [TCSEC]), all storage-based information flows at the interface (e.g., inputs, return values and error messages) are represented, typically as changes to state variables. Since all of the interface flows are represented and covert storage channels are signalled through a change to a storage location at the interface, the formal covert channel analysis of a complete and accurate FTLS is assured of revealing the covert storage channels of the system represented.

On the other hand, the type of delays that drive a timing channel are not specified in a DTLS or FTLS using current specification and verification methods [HAIGH86, p. 17]. Thus, unlike covert storage channels, covert timing channels cannot be identified from an FTLS but must be identified informally by a careful examination of system internals.

Description of Approach

The covert channel analysis of the GEMSOS Kernel utilized the FDM tool set. Included in this set are the Ina Jo specification language and processor [SCHEI88], the Ina Flow tool [ECKM87] (including the MLS flow theorem generator and the SRM matrix generator) and the Interactive Theorem Prover (ITP)[SCHOR88].

Theoretical Approach

The FDM tools are designed to be used in the following general method to analyze information flow in a system [ECKM87]:

1. Describe the system interface in the Ina Jo Specification Language in terms of exceptions, error messages and effects. Use the Ina Jo processor to check the syntax of the specification.
2. Define security labels for all variables within the specification.
3. Produce flow theorems from the labeled specification using the MLS tool.
4. Prove flow theorems using the ITP (unproven theorems are theoretical, "formal," flow violations).
5. The exploitability of theoretical flow violations is determined manually.

Alternatively, the SRM tool produces a "Shared Resource Matrix" (as defined by Kemmerer [KEMM83]) from an input specification. The matrix lists all transforms (representing system functions that can produce state changes) and variables, and shows whether a variable is read or modified in each transform. The tool output includes a transitive closure of the references [1]. Finally, the legality of flows and the exploitability of flow violations are determined.

1. Reference transitivity is illustrated with two transforms and three variables (V1, V2 and V3). One transform reads V1 and writes V2. The second transform reads V2 and writes V3. Information flows transitively from V1 to V3, via V2. The output from the SRM tool would show that the second transform reads V1. A transitive closure of references provides all of the references derivable through the transitivity of information flow.

Actual Approach

In the covert storage channel analysis of the GEMSOS Kernel, the first two steps of the MLS theoretical approach were completed and the labeled specification was processed with the MLS flow tool. For various reasons owing to the immaturity of the tools at the time, they were unable to correctly process the specification. For example, MLS had difficulty with non-determinism and the Ina Jo language did not allow structure fields to be included in the label ("clearance") statements (see example below).

An alternative approach of working from a Shared Resource Matrix derived from the specification was investigated. It was found that the SRM tool at that time could not generate a single matrix for all of the transforms due to the size of the specification. After some experimentation it was determined that the tool was able to generate the matrix one column (i.e., transform) at a time. This discovery lead to a closer look at utilizing the Shared Resource Matrix methodology.

A problem with the SRM approach was our lack of access to a tool that could generate the required transitive closure of references (i.e., the SRM tool could only deal with one transform at a time). Performing the transitive closure by hand was considered beyond the scope of effort for the project. After this problem was resolved (see "Transitive Closure," below), we defined an approach which combined the methods of SRM and flow analysis [DENN76, MILL76]. First, we used the SRM tool to detect all of the variable references (read, write) generated within a transform. Next, we labeled the variables, and performed a semantic analysis of the context of the references within the specification to detect illegal flows. This analysis included the criteria identified by Kemmerer to determine the suitability of the flows as covert channels. Finally, to help determine the fastest way to drive the channels, a reduced SRM was produced (see "Matrix Reduction," below).

Transitive Closure

Transitive closure of the flows in the shared resource matrix is normally provided by the Ina Jo tool used to create the matrix. Since our matrix was created by hand, the issue of transitive closure was considered independently. We determined, much as did Tsai [TSAI87], that transitive closure was not necessary. It was clear that transitive closure would only provide illegal flows based upon other already known direct illegal flows (See Appendix for a formal proof of this property).

The point of covert channel analysis is to identify information leakage such that it may be limited (in the best case, closed) and/or audited. In the case of audit, since each transitive flow utilizes one or more direct illegal flows, the usage of each transitive channel will trigger the audit mechanism for its direct flow(s). For the limitation of transitive-flow based channels, we concluded that since the transitive flows result from a serial concatenation of direct flows [2], the overall transitive channel could not operate any faster than the direct flows upon which they were based. Thus, limitation and audit strategies for a direct channel will similarly limit and provide audit for its associated transitive-flow based channels.

If the direct illegal flows of a system are both auditable and limitable, the only obvious benefit to performing transitive closure is if a direct illegal flow is dismissed as unusable (i.e., not considered a covert channel) because a variable involved could not be seen directly or manipulated at the interface. If this rationale were used for elimination of a possible channel then it seems that one would be forced to analyze the transitive closure on the matrix before reaching the conclusion that the illegal flow is unusable. Since we did not eliminate any illegal flows this way, the requirement for transitive closure was obviated.

2. Note that the transitive flows discussed herein utilize the serial concatenation of flows to produce a channel, whereas channel aggregation [TSAI88, p. 113] refers to the parallel and symbiotic exploitation of different covert channels.

For the GEMSOS Kernel, we found that the direct illegal flows were both auditable and limitable. The measurement of direct flows also provides input, in conjunction with knowledge of system configuration information, to perform various sorts of channel aggregation measurements should such measurements be desired.

Generation of Variable References

Each transform of the specification was run through the SRM tool. This generated a list of references for each transform, somewhat like the following partial output for a transform (swapin_segment) for moving data from secondary storage into main memory.

```
        T6               KEY
    ---------
    V1 | RM         V1 :  proc_table(pid).mem_avail
    V2 | RM         V2 :  a_table(pid, sn).swapped_in
    V3 | RM         V3 :  global_mem_avail
    V4 | M          V4 :  success(pid)
                    T6 :  swapin_segment
```

Shown is an SRM with one transform and four variables, along with a key to the transform and variables. In the SRM, "R" indicates read and "M" indicates modify; "pid" is an identifier of type process ID, "sn" is an identifier of type process_local_segment_number.

Labeling of Variables and Semantic Analysis

After the lists of references within each transform was generated, the variables were labeled. We developed the following conventions for this process:

1. All constants (i.e., variables that were only read but never written) were labeled "sys-low"
2. All variables that were read by all processes were labeled "sys-low"
3. All variables that were written to by all processes were labeled "sys-hi"
4. All variables that were indexed by process were labeled "at the process level" which we assumed to be in the range sys-hi to sys-lo
5. All variables that were both written and read by all processes were labeled "syshi." Note that it doesn't matter whether the bidirectional illegal flows are considered bad reads or bad writes since either way they are flagged as potential contributors to covert channels.

The variable's labels were compiled in a global list, such that each variable was treated consistently across all of the transforms. Examples of the variable labels are shown below in the syntax of Ina Jo. A variable to the left of an "at" sign is assigned the label to the right of the "at" sign. The function "sec_label" returns a label for the process ID argument (pid).

```
a_table(pid, sn) @ sec_label(pid),
global_mem_avail @ syshi,
proc_table(pid) @ sec_label(pid),
success(pid) @ sec_label(pid)
```

A semantic analysis of each flow identified by the SRM tool was performed. This analysis was done by hand due to the immaturity of the flow tool. The semantic analysis of the references was documented in a list which gave a brief rationale for the outcome. Usually, the analysis involved comparing the process and variable labels directly. In some cases a more detailed rationale was required, such as relying on system invariants or explicit security checks in the specification to infer the relationship of the process and variable labels; these rationales were formulated as closed deductive arguments. The following rationales reflect a security policy for single-level processes which requires a process to be at or above the level of an observed object, and at or below the level of a modified object.

V1 I RM I legal because proc_table(pid) is at level of pid
V2 I RM I legal because a_table(pid,sn) is at level of pid
V3 I RM I ILLEGAL because global_mem_avail is at system high
V4 I M I legal because success(pid) is at level of pid

The semantic analysis included meeting the following requirements to be the source of covert channels [KEMM83]:

1. Sending and receiving processes must be able to access the same attribute of a shared resource.
2. The sending process must be able to write to the shared attribute.
3. The receiving process must be able to read the shared attribute.
4. There must be some mechanism for initiating the sending and receiving processes and for sequencing the events correctly.
5. The sending and receiving processes must be in distinct protection domains and must not be allowed to communicate with each other directly.

Matrix Reduction

A matrix was created consisting of all variables involving direct illegal references, and all transforms with references to those variables.

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
gast_total		R		Rm				m	rm	Rm
global_mem_avail		rm				Rm	m	m	rm	Rm
last_total		R		Rm				m	rm	Rm
local_mem_avail		rm				Rm	m	m	rm	Rm
total_active_processes									rm	Rm
total_mounted_volumes			rm		Rm					
vol_space_avail	R	rm			Rm					

r = read
R = illegal read
m = modify

The reduced matrix had 10 transforms and 7 variables. This is in contrast to the output of the SRM tool, which would have shown 30 transforms and 738 variables and constants. A similar reduction in the number of references (e.g., r, m) recorded is also apparent.

As explained above and in the Appendix, the excluded variables and transforms do not need to be included in the covert channel analysis: any operations that indirectly reference a variable are not of interest because the auditing and reduction of the covert channels is accomplished relative to the direct illegal reference.

We have found that the reduced matrix provides significant information necessary for covert analysis of the system. A covert channel involves complementary actions: reading a variable in question and writing the variable. The reduced matrix includes all direct illegal references and shows all of the transforms that can be utilized in the complementary action to each illegal reference. For example, in the case of the above matrix where all of the illegal references are illegal reads, one can determine which operations can be used to directly write to the variable. This information can be used in bandwidth estimation (see "Channel Bandwidth Estimation," below) as well as limiting and auditing of the channel.

The reduced matrix is a subset of the full transitive closure matrix. This will be true in general since a transitive closure matrix is an expansion of a matrix of direct flows, and a reduced matrix takes as input a direct flow matrix, and reduces it (by eliminating variables without illegal references).

The entries in the matrix were then analyzed to determine the best scenario for exploitation of the illegal flows in the form of covert channels.

Channel Bandwidth Estimation

The analysis of illegal direct flows revealed that they were primarily resource exhaustion channels. The one exception was considered a design flaw. Security checks were added to the kernel interface to eliminate this channel, and the analysis was adjusted accordingly. The resource exhaustion channels were found to be closeable through proper system configuration choices and were all auditable. However, in order to provide customers with a basis for deciding if the restrictions imposed by configuration options were necessary, analysis was performed to estimate the maximum theoretical bandwidth of each of the channels.

In some cases, a single covert channel (relative to a system variable) could be exercised through multiple pairs (reader and writer) of kernel calls (see the matrix, above). In order to determine which of these pairs would provide the highest estimated bandwidth, the speed of each kernel call was tested. The fastest pair that exercised a given channel, based on those listed in the matrix, was then used in the estimation of the channel's bandwidth.

The actual bandwidth estimates and exploitation scenarios resulting from this analysis are proprietary and are not included in this report.

Informal Identification of Covert Storage Channels

In a separate effort from the formal covert storage channel analysis based on the FTLS, an informal engineering analysis of the DTLS was performed. This separate analysis involved the evaluation of the order of outputs described in the DTLS to determine whether the outputs represent illegal flows and could be used for covert channel exploitation. The relevance of the "output ordering" analysis to the covert channel analysis is based on the assumption that all illegal flows are detected at the interface through outputs returned by the kernel. The illegal flows thus discovered corresponded to the illegal reads identified in the SRM matrix, above.

The analysis method is particularly applicable for systems below the class A1 level where an FTLS and the associated formal analysis are not available. At the A1 level, the informal analysis can provide a useful counterpoint to, and a further validation of, the formal analysis. Although the informal analysis is necessarily less reliable than formal analysis, it was far less time consuming.

Description of Approach

For this analysis, the DTLS has the following characteristics:

1. All state variables are identified as "process-local" or "global."
2. The security level of each state variable is identified (the conventions used are as described for the formal analysis).
3. For each output, the state variables that are observed in order to return the output are identified.

Outputs are either error messages (indicating exception conditions) or return values. The return of an output by the kernel typically indicates observation of one or more state variables (i.e., attributes) within the kernel. "Process-local" state variables are observed and modified by a single process only. Outputs returned to a process as a result of observation of "process-local" state are legal since the information is at the same level as the process.

"Global" state variables are observed and modified by more than one process. An output returned to a process as a result of observation of a "global" state variable may be part of an illegal flow. For each kernel output so identified, an ordering analysis is performed to confirm that the design prevents the illegal flow.

In the GEMSOS Kernel, outputs are ordered: in the event of an exception, only an error message is returned as an output; the order in which exception conditions are checked determines the order of the their corresponding outputs; in the event of two or more exceptions, only the condition that is checked first will be reflected as output.

Each output associated with the observation of a global state variable must be ordered to occur AFTER a corresponding output representing a system security check (the specific checks are described below). If a "global-observing" output is out of order with respect its corresponding system security check, or the check is absent, then a covert channel is identified.

The outputs were divided into two classes for this analysis: those indicating global resource exhaustion, and "other." For global resource exhaustion, the corresponding system security check determines whether the process-local allocation of the resource is exhausted. This ordering reflects the kernel mechanism for partitioning global resources on a per-process basis, such that with proper system configuration (i.e., initial allocation), a process will always exhaust its local resource allocation before exhausting the global resource.

For outputs other than global resource exhaustion exceptions, (for example, the return of file size), the corresponding system security check must confirm that the calling process is at a security level sufficient to observe the global state.

Conclusion

Although the tools exist today for performing analysis of specifications with respect to flows and covert channels, these tools are not of sufficient maturity to be used effectively in the automated analysis of an commercially-sized operating system kernel. We have shown that it is feasible to work with the currently evolving tools and complete a formal covert channel analysis on a relatively large specification. Informal "output ordering" analysis yielded results that were consistent with the formal covert channel analysis results.

We chose to base our analysis on the direct illegal flows rather than on the transitive closure of flows because: 1) direct illegal flows are the fundamental leakages of the system (all illegal flows evolve from direct illegal flows); 2) we were able to address (audit and limit) those flows directly; and, 3) we wanted to limit the level of effort of the analysis.

By adapting the analysis methodology to the capabilities of the tools and methods available today, one can arrive at a significant reduction in effort relative to theoretical covert channel analysis approaches. The methodology outlined here presents a viable alternative for use while analysis tools mature. The authors recommend continued research and development in automated analysis systems. It is hoped that the techniques introduced here to reduce the necessary amount of analysis can be incorporated into future tools.

Acknowledgements

The authors wish to express thanks to Tim Redmond of TIS for his contributions to the appendix on transitive closure and to Mark Heckman and the National Computer Security Conference referees for their insightful review comments.

References

[DENN76] D. Denning, "A Lattice Model of Secure Information Flow," in *Communications of the ACM*, pages 236-243, ACM, May 1976

[ECKM87] Steven T. Eckmann, "Ina Flo: The FDM Flow Tool," in *Proceedings of the Tenth National Computer Security Conference*, pages 175-182, National Bureau of Standards/National Computer Security Center, 1987 Gaithersberg, MD

[HAIGH86] J. Haigh, R. Kemmerer, J. McHugh, W. Young, "An Experience Using Two Covert Channel Analysis Techniques On a Real System Design," in *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland California, 1986

[KEMM83] R. Kemmerer, "The Shared Resource Methodology: An Approach to Identifying Storage and Timing Channels," *ACM Transactions on Computer Systems*, pages 256-277, August 1983, University of California, Santa Barbara

[MILL76] J. Millen, "Security Kernel Validation in Practice," in *Communications of the ACM*, pages 243-250, ACM, May 1976

[SCHEI88] J. Scheid and S. Holtsberg, *The Ina Jo Specification Language Reference Manual*, Unisys Corporation, 2400 Colorado Ave, Santa Monica CA 90406-9988, 1988

[SCHEL85] R. Schell, T.F. Tao, and M. Heckman, "Designing the GEMSOS Security Kernel for Security and Performance", in *Proceedings of the Eighth National Computer Security Conference*, Gaithersberg, MD, October 1985, pp. 108-119

[SCHOR88] V. Schorre, et. al., *The Interactive Theorem Prover Reference Manual*, TM 6889/000/08, 10 November 1988

[TCSEC] *Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, December 1985

[THOM90] M. Thompson, R. Schell, A. Tao, T. Levin, "Introduction to the Gemini Trusted Network Processor," in *Proceedings of the 13th National Computer Security Conference*, Gaithersberg, MD, 1990

[TNI87] *Trusted Network Interpretation of Trusted Computer System Evaluation Criteria*, NCSC-TG-005 Version-1, 31 July 1987

[TSAI87] C.R. Tsai, "Covert-Channel Analysis in Secure Computer Systems," Phd. Dissertation, University of Maryland, College Park, Maryland, August 1987

[TSAI88] C.R. Tsai, V. Gligor, "A Bandwidth Computation Model for Covert Storage Channels and Its Applications," in *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland California, 1988

Appendix: Proof of Transitive Closure

This appendix provides a proof that no illegal flows will be created by taking the transitive closure on a shared resource matrix that has no illegal direct flows. This shows that if one eliminates the direct illegal flows from an SRM, the transitive closure will introduce no new illegal flows. Therefore, if there exist illegal flows in the transitive closure of an SRM, they are derived from the illegal direct flows in the base SRM. The proof is trivial but is included for completeness.

We begin by defining:

T = finite set of all transforms (fixed for appendix)
V = finite set of all variables (fixed for appendix)

Fix atoms, R and M, intuitively denoting the notions of read and modify. Fix a set of labels and a partial ordering relation on this set, "<." Fix a function, "label," which maps elements of V to elements of the set of labels.

DEFINITION 1. A shared resource matrix, F is a matrix indexed by T x V such that for all t in T, v in V: F(t,v) is a subset of {R,M}. We will use F, F' etc., to denote shared resource matrices.

DEFINITION 2. A flow for a shared resource matrix, F, is a triple, (t,v1,v2) where R is an element of F(t,v1) and M is an element of F(t,v2). We will also denote the flow (t,v1,v2) as (v1 t-> v2).

DEFINITION 3. A flow (v1 t-> v2) is said to be a legal flow iff label(v1) < label(v2).

DEFINITION 4. A "contains" relation which provides a partial ordering on shared resource matrices is defined such that F' contains F iff for all t in T, v in V: F(t,v) is a subset of F'(t,v).

DEFINITION 5. A shared resource matrix, F, is transitively closed iff for all t1 and t2 which are elements of T, v1 and v2 which are elements of V: [R is an element of F(t1,v1) and M is an element of F(t1,v2) and R is an element of F(t2,v2)] implies [R is an element of F(t2,v1)].

DEFINITION 6. If F is a shared resource matrix, then F" is the least shared resource matrix that contains F and is transitively closed.

The construction of F" is typically performed in steps. These steps will be called transitive closure steps. A transitive closure step takes F to F' if there exists a t1, t2, v1, v2, such that:

> R is an element of F(t1,v1) and M is an element of F(t1,v2) and R is an element of F'(t2,v2) and R is not an element of F(t2,v1)
>
> and for all t which are elements of T, v which are elements of V: [t not equal t2 or v not equal v1] implies [F(t,v) = F'(t,v)].
>
> and F'(t2,v1) = (F(t2,v1) Union (r))

If one begins with a resource matrix and repeatedly applies the transitive closure step until no more transitive closure steps can be applied to the matrix then the resulting matrix is transitively closed.

THEOREM

If all the flows for F are legal then all the flows for F" are legal.

PROOF

Suppose a sequence F0, F1, ..., Fn where each F is a transitive closure step of the previous F, and Fn = F", and F0 = F. We will show by induction on k that all flows in Fk are legal. For k=0 this is obvious.

Suppose there exists a t1, t2, v1, v2, v3 which are appropriate for some F(k-1) to Fk and all flows in K-1 are legal (see next figure). All flows for Fk that are not flows for F(k-1) are of the form: (v1 t2-> v3). It is easy to see that (v2 t2-> v3) is a flow for F(k-1), so label(v2) < label(v3).

```
F(k-1)       V1  V2  V3
         T1  r   m         (v1 t1-> v2)
         T2      r   m     (v2 t2-> v3)

F(k)         V1  V2  V3
         T1  r   m         (v1 t1-> v2)
         T2  r   r   m     (v2 t2-> v3)
                           (v1 t2-> v3)
```

Example Matrices

Also, (v1 t1-> v2) is a flow of F(k-1), so label(v1) < label(v2).
By transitivity on <: [label(v2) < label(v3) and label(v1) < label(v2)] implies [label(v1) < label(v3)].
Which means that the flow is legal and all flows in Fk are legal.
Q.E.D.

Verification of the C/30 Microcode Using the State Delta Verification System (SDVS)[1]

Jeffrey V. Cook

The Aerospace Corporation
P. O. Box 92957
Los Angeles, CA 90009

Abstract

We present the formal verification, using the State Delta Verification System (SDVS), of the microcode for the Bolt Beranek and Newman, Inc. (BBN) C/30 computer. The C/30 has a high-level instruction set architecture that is emulated by microcode resident on BBN's Microprogrammable Building Block (MBB) computer. A large majority of the C/30's instructions were proven to be correctly emulated, but some microcode errors were discovered during the verification process. This verification effort, which demonstrated SDVS' ability to check the correctness of microcoded computer implementations, is a significant milestone on the path to correctness proofs that span the hardware/firmware/software hierarchy.

1 Introduction

This paper describes the C/30 Microcode Verification Project, which was initiated at The Aerospace Corporation in October 1984 and was completed there in November 1986. The project involved formally proving the correctness of microcode that emulates the instruction set architecture of the C/30 computer. The C/30 computer [1], designed by Bolt Beranek and Newman, Inc. (BBN), was implemented by microcode for BBN's Microprogrammable Building Block (MBB) [2, 3]. The proof of microcode correctness was specified and verified using the State Delta Verification System (SDVS) [4], a system developed at The Aerospace Corporation. SDVS is a system for writing, and checking the correctness of, proofs of statements written in its internal temporal logic, the *state delta logic* [5].

The C/30 Microcode Verification Project was of major significance for at least two reasons. First, the MBB is a production computer, not a toy computer, for which the emulation of the C/30 architecture is only one of its many uses. The C/30 has been in operation for many years as a packet switching node[2] on the Arpanet. The second significant aspect was the amount of microcode involved. Approximately 1000 MBB microinstructions implemented the portion of the C/30 instruction set that was verified during the project. A large majority of the C/30's instructions were proven to be correctly implemented by the microcode, but a number of microcode errors were discovered during the verification process.

Two other significant hardware and microcode verification efforts have been undertaken in recent years. One consisted of the use of the HOL system to verify the correctness of the Viper microprocessor in 1987 [6, 7, 8]. Another consisted of the use of the Boyer-Moore system to verify the correctness of the FM8501 in 1986 [9, 10].

[1]This research was supported by the National Computer Security Center under contracts FO4701-83-C-0084 and FO4701-85-C-0086.

[2]The terminology "IMP," or "interface message processor," may be more familiar to some readers, as it predates "packet switching node."

SDVS is briefly discussed in Section 2, followed by a discussion of SDVS's microcode verification paradigm in Section 3. The MBB and C/30 computers are described in Section 4. The formal specifications of the architectures of these two computers are described in Section 5. The formal statement that the microcoded MBB correctly implements the C/30 is given in Section 6. The proof of this statement of implementation correctness is discussed in Section 7. Finally, Section 8 concludes this paper with observations concerning the verification process.

2 SDVS and State Deltas

A good general introduction to SDVS is given in [11], even though some information specific to an older version of SDVS is found there. Reference [12] is the *SDVS Users' Manual* in effect at the time the C/30 Microcode Verification Project was completed. A recent paper that describes SDVS, state deltas, and the translator for a subset of Ada[3] is given in [13]; most of the material in this section is taken from this paper.

SDVS is a system for checking proofs about the course of a computation. SDVS is based on a specialized form of temporal logic whose temporal formulas are called *state deltas*. A state delta is a description of a transition from one computation state to another. Its *precondition* describes a state from which the transition can be made, and its *postcondition* describes the state resulting from the transition. Technically, SDVS checks proofs of state deltas, which provide an operational semantic representation of computation. SDVS can handle proofs of claims of the form, "if P is true now, then Q will become true in the future." If P is a program (perhaps with some initial assertions) and Q is an output assertion, then the above claim is an input-output assertion about P. SDVS can also handle claims of the form "if P is true now, then Q is true now."[4] In this case, if P is a program and Q is a specification, then the claim asserts the total correctness of P with respect to Q. SDVS is also capable of handling proofs that one computer program (or description) correctly implements another, i.e., *multilevel* correctness proofs.

A state delta is a formula consisting of a precondition P, a comodification list C, a modification list M, and a postcondition Q. P and Q are non-empty lists of formulas taken from the language of the state delta logic. C and M are (possibly empty) lists of *places*. A *place* contains (abstract) *values*, the place's "contents." Places can be viewed as, for example, abstract memory locations or program variables. SDVS displays state deltas using the following notation:

```
[SD pre: P
  comod: C
    mod: M
   post: Q ]
```

Let the times t_1 and t_2 denote a state delta's precondition and postcondition times, respectively. A state delta's modification list M specifies those places whose contents are allowed to change between precondition and postcondition time as a result of the transition. The truth value of any assertion about these places *cannot be assumed to be preserved* during the transition. The contents of places not listed in the modification list *must remain unchanged* during the state transition. State deltas assert the *total correctness* (in the Floyd-Hoare sense) of programs whose transitional behavior they characterize with respect to the state delta pre- and postconditions (together with

[3]Ada is a registered trademark of the U. S. Government – Ada Joint Program Office.

[4]In addition, SDVS can handle claims of the form "for every time in the future Q is true" for arbitrary predicates Q.

the implicit assertions that the places not in state delta modification lists preserve their contents across the associated state transitions). The role of a state delta's comodification list C is more subtle and is explained in detail in [13].

Note that SDVS is not only a system for *checking* the correctness of proofs, but it is also a system for interactively *developing* proofs. A user may interactively guide SDVS's proof-checker with high-level proof commands (e.g. symbolically execute, induct, prove by cases), while many low-level deductions are made automatically. In particular, SDVS contains decision procedures for the theories of propositional logic and equality between uninterpreted function symbols, and *partial* decision procedures for the theory of Presburger arithmetic, a theory of arrays, and a theory of bitstrings, among others.

3 Microcode Verification

In this section we discuss the microcode verification paradigm of SDVS, and then relate it to the C/30 Microcode Verification Project. This paradigm entails proving that the instruction set architecture (ISA) of a virtual computer is correctly emulated by a microcoded computer. We shall use the terms *emulated* and *microcoded* to refer to these two computers, respectively. Proofs in this category are referred to as proofs of *implementation correctness* [14].

In order to prove properties of a computer, SDVS requires a formal description of that computer. When the C/30 Microcode Verification Project was initiated in 1985, the only hardware description language recognized by SDVS was ISPS (Instruction Set Processor Specification), described in [15]; ISPS had been in use for over a decade as a language for describing hardware at the register transfer level. A translator was developed and implemented for a nontrivial subset of ISPS. This translator converts ISPS statements into state deltas and other logical formulas. Thus, SDVS has the capability to prove correctness properties of computers described in the accepted subset of ISPS.

In addition to the ISPS descriptions, two other items are necessary to construct the statement of implementation correctness: the constants of the microcoded computer (such as its microcode), and a formal mapping from the emulated computer to the microcoded computer. This mapping shows the relationships between states and storage locations in the two machines.

The microcode verification paradigm for the C/30 is shown in Figure 1. The Microprogrammable Building Block (MBB) emulates the instruction set architecture (ISA) of the C/30 via a microprogram tailored for that purpose. We refer to this microprogram as the *C/30 Microcode*; the proof of implementation correctness for this microcode is referred to as the *C/30 Proof.* As shown in the figure, the user provides the ISPS descriptions of the C/30 and of the MBB, a formal mapping between the two machines, and the actual binary microcode for the C/30. From these are constructed the statement of implementation correctness, designated the *C/30 State Delta.* The two inputs to SDVS are the C/30 State Delta and the C/30 Proof.

Although Figure 1 has been greatly simplified for the purposes of this discussion, we emphasize that the verification process was a task of considerable magnitude. For the C/30 State Delta to be constructed, the ISPS descriptions of the C/30 and the MBB had to be written and the mapping between the states and registers of both machines had to be determined. Only then could we begin to develop and verify the C/30 Proof using SDVS, which required a high degree of interaction between the author and the proof system.

For complicated computers, the development and verification of such a proof is an arduous

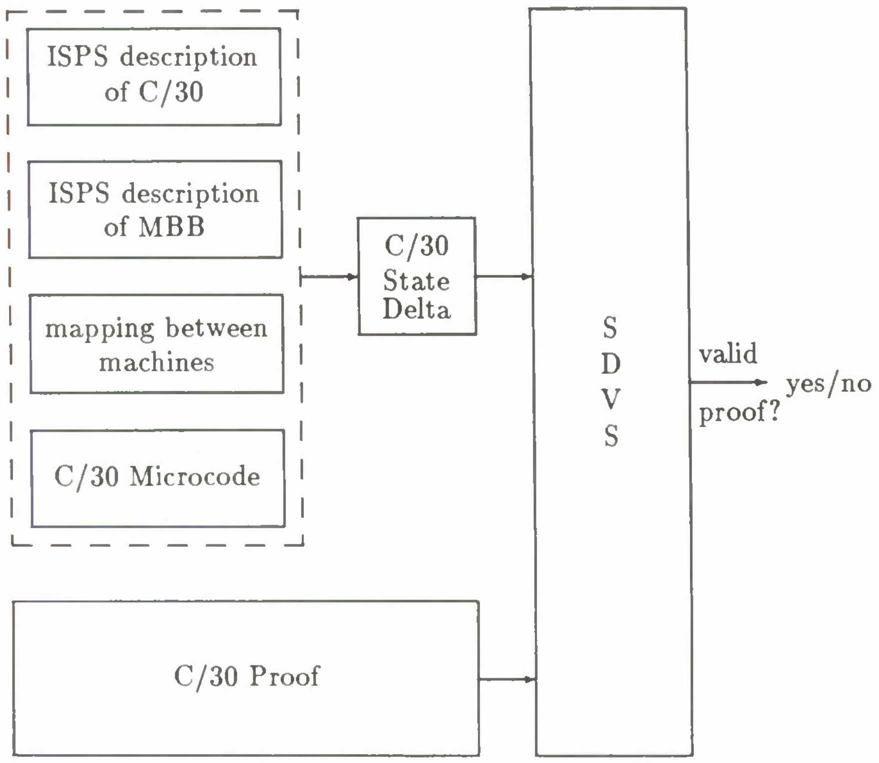


Figure 1: C/30 Microcode Verification using SDVS

process, requiring an in-depth understanding of the microcoded computer, its microcode, and the emulated computer. If an emulated computer instruction is improperly microcoded, no correctness proof can be achieved.

One utility of microcode verification is demonstrated when the verification process uncovers microcode errors. Of course, the gross errors are the more easily recognized, and are usually uncovered by machine-language programmers when certain microcoded machine-language instructions are discovered to operate incorrectly. If an erroneous instruction is not crucial, that is, if its operation can be implemented by some other combination of instructions, then the machine-language programmer must bypass the erroneous instruction until the microcode is fixed. Thus subtle microcode errors may or may not be discovered by machine-language programmers, and may lie in wait for years before causing a serious program malfunction.

4 The MBB and the C/30

As noted above, the Microprogrammable Building Block (MBB) emulates the instruction set architecture (ISA) of the C/30 via the C/30 Microcode. The C/30 was chosen for verification because of interest in the verification of certain aspects of the Defense Data Network (DDN), and because of the existence of a formal ISPS description of a version of the MBB.

The MBB is a general-purpose microprogrammable computer that can be used for a variety of applications. The MBB's main purpose, as envisioned by the designers, is to emulate other computers. In particular, it is capable of emulating the ISA of the C/30. For each computer emulated, the MBB requires the insertion of two custom-designed "daughter" boards, the MIRDB

(Macroinstruction Registers Daughter Board) and the MARDB (Memory Address Register Daughter Board).

The C/30, specifically designed to serve as a packet switching node on the DDN, is one of a family of computers developed by BBN. The C/30 is a 16-bit/word machine with 64K words of addressable memory and three addressing modes. It has a number of special-purpose and general-purpose registers, and a set of 128 instructions, including sophisticated instructions for manipulating queue data structures and controlling multiprocessing. It operates a polled interrupt system with clock, I/O, and scheduling interrupts.

5 Formally Specifying the MBB and the C/30

In this section we discuss the ISPS descriptions of the MBB and the C/30. A discussion of the problems that arose from the use of ISPS as a hardware description language are presented in [16] and [17].

5.1 ISPS Description of the MBB

The C/30 Microcode Verification Project took advantage of an existing description of another machine, the C/70 MBB [18]. Converting the ISPS description of the C/70 MBB into an ISPS description of the C/30 MBB required changing two components of the C/70 MBB description, the ISPS descriptions of the MIRDB and the MARDB. In addition, the size of the main memory of the MBB was reduced from 1M to 64K. The ISPS description of the C/30 MBB is given in [19], with commentary on the computer's operation. This ISPS description occupies 30 pages of text, or 15 pages in the absence of text formatting.

A portion of the C/70 MBB description that was excised before the C/30 Proof began was that of the error detection and correction (EDAC) algorithm that checks for data errors during main memory reads. Thus, the C/30 Proof assumes that no data errors (e.g. parity errors) occur during main memory reads. Henceforth, the term "MBB" shall refer solely to the C/30 configuration of the MBB computer.

5.2 ISPS Description of the C/30

The ISPS description of the C/30 computer [20] was written from documentation supplied by the *C/30 Programmer's Reference Manual* [1], and from interactions with BBN employees involved in the C/30 Microcode Verification effort. Ten of the 128 instructions in the C/30 instruction set were not included in this description. These ten included instructions that manipulate the I/O system of the C/30, whose actions were difficult to specify formally, and the maintenance and diagnostic instructions, which had the capability of altering the C/30 Microcode (the C/30 Microcode was assumed to remain unchanged during the C/30 Proof). This ISPS description occupies 41 pages of text, or 17 pages in the absence of text formatting.

6 The Statement of Implementation Correctness

Once the ISPS descriptions of the MBB and the C/30 were available, the formal statement of implementation correctness for the C/30 could be constructed. Let **c30micro.isp** denote the name of the file containing the ISPS description of the MBB, and let **c30macro.isp** denote the name of the file containing the ISPS description of the C/30. The notations **isps(c30micro.isp)** and `mpisps(c30macro.isp)` represent state delta translations of these descriptions; these notations are discussed in more detail below. The statement of implementation correctness for the C/30 is then represented in SDVS by the state delta shown below. (This is a stylized, abbreviated representation of the actual C/30 State Delta; italics have been used to represent missing formulas.)

```
[SD pre: (isps(c30micro.isp) ∧
          MBB constants, e.g., the C/30 Microcode ∧
          mapping from C/30 to the MBB)
  comod: ()
    mod: ()
   post: (mpisps(c30macro.isp)) ]
```

Informally, this state delta says that the MBB computer, with the C/30 Microcode and certain other constants, implements the C/30 computer, via a mapping that relates the states and architectures of the two computers. The exact statement of implementation correctness for the C/30 is given in [21].

The two unary SDVS predicates `isps` and `mpisps` are used to capture the semantic output of the ISPS translator as follows. The formula `isps(c30micro.isp)` denotes the *incremental* translation of the ISPS description of the MBB. This predicate is useful only for the symbolic execution of ISPS descriptions, because it incrementally translates ISPS descriptions one statement at a time. The notation `mpisps(c30macro.isp)` denotes the mark-point to mark-point[5] translation of the ISPS description of the C/30. This predicate is useful when one wishes to prove properties (such as correct implementation) of an ISPS description of a computer. The `mpisps` translation yields a set of logical formulas that describe the static architecture of the emulated computer, as well as a set of state deltas, one state delta for each possible execution path between successive labels in the ISPS description.

7 C/30 Proof

The primary purpose of the C/30 Microcode Verification Project was to produce a verified proof of correctness of the C/30 Microcode. This section discusses the portions of the C/30 ISA not verified by the C/30 Proof, some of the strategy for the C/30 Proof, a summary of the proof, and the C/30 Microcode errors discovered during the verification process.

7.1 C/30 Proof Omissions

For reasons discussed briefly below, the complete verification of certain C/30 instructions was not attempted. Full details are supplied in [22].

Certain long-running C/30 instructions, in particular the shift instructions and the **CCRO** (Convert and Clear Rightmost One) instruction, are interruptible by the clock and I/O interrupts. These instructions were verified under the assumption that no interrupts occurred during their execution, because the exact method and timing of their interruptibility were not documented, and because in 1985 SDVS lacked capabilities for modeling their interruptibility in a way that was independent of a specific implementation.

[5]ISPS labels are mark-points. SDVS introduces implicit mark-points to label the beginning and end of ISPS descriptions.

However, the interruptibility of four block-transfer and block-checksum instructions (**BLT**, **TRB**, **CHK**, and **ECK**), whose interruptibility was explicitly mentioned in the documentation, was modeled in the ISPS description of the C/30. Their interruptibility was modeled in a manner dependent on the C/30 Microcode implementation; this permitted the development of correctness proofs for these four instructions.

Time constraints and difficulties in accurately modeling certain aspects of the C/30 architecture prevented the verification of four multiprocessing instructions (**NMFS**, **DPR**, **SPR**, and **GPR**), and resulted in only a partial verification for one multiprocessing instruction (**ENB**). In addition, because of time constraints alone, the actions of the clock interrupt and the programmable (multiprocess scheduling) interrupt were not verified. The difficulties in modeling were due to the complexity of the instructions involved and incomplete documentation of their operation.

7.2 C/30 Proof Strategy

The strategy for developing the C/30 Proof is the topic of another report [23]. The actual text of the proof and the theorems proved during the verification of this proof appear in [21].

To prove the truth of the C/30 State Delta, one must prove the truth of the formulas denoted by `mpisps(c30macro.isp)`. In the ISPS description of the C/30, the label `c30macrocycle` marks the beginning of the C/30 instruction-interpretation loop. In this particular description, it also marks the end of the loop, because execution returns to the label after each iteration. Thus, the contents of some of the state deltas denoted by the predicate `mpisps(c30macro.isp)` are determined by the execution paths within the C/30 instruction-interpretation loop, with the label `c30macrocycle` delimiting the beginning and endpoints of each of these state deltas.

The proof process is best illustrated by an example. Consider the C/30 instruction **IAB** (Interchange A and B registers). The 16-bit binary operation code for this instruction is 0000000010000001, or 129_{10}. An abbreviated representation of the state delta describing the actions of **IAB**, derived directly from the set of state deltas denoted by the predicate `mpisps(c30macro.isp)`, is shown below. (Note that while italics are used to represent missing formulas, ellipses are used to represent missing or irrelevant portions of the state delta.)

```
[SD pre: (at label c30macrocycle in ISPS desc. of C/30 ∧
          .MEM[|.PC|]=129(16) ∧
          ...)
  comod: (...)
    mod: (A,B,PC,...)
   post: (at label c30macrocycle in ISPS desc. of C/30 ∧
          #A = .B ∧ #B = .A ∧
          #PC = (.PC ++ 1(2))<15:0>) ∧
          ...) ]
```

Let IABSD denote the above state delta. IABSD's precondition states that the C/30 is at the beginning of its instruction-interpretation cycle and the operation code of the current instruction has the value 129; its modification list permits changes to the A and B registers, and to the program counter (PC); and its postcondition states that the C/30 is once again at the beginning of the instruction-interpretation cycle, the contents of the A and B registers have been swapped, and the content of the PC register has been incremented by 1, modulo 2^{16}.

The primary objective of the C/30 Proof is to prove the C/30 State Delta, which contains a representation of IABSD in its postcondition. To prove IABSD, under the assumption that the C/30 State Delta's precondition holds (the ISPS description of the MBB is available for symbolic execution, the C/30 Microcode has a certain value, and a mapping holds between the C/30 and the MBB), one must perform the following steps:

1. Assert the truth of the IABSD precondition.
2. Symbolically execute the ISPS description of the MBB.
3. Determine if the IABSD postcondition holds.

The mapping is used to map C/30 states onto MBB states, and to map C/30 registers (such as A and B) onto MBB registers. Mapping the IABSD precondition results in the positioning of the MBB's state at the top of its microinstruction-interpretation loop, at the point where the next C/30 instruction is to be emulated; it also ensures that the proper operation-code value is in the memory location of the instruction to be emulated. One then symbolically executes state deltas from the translation of the MBB description; this process interprets the binary microcode that comprises the microroutine for the **IAB** instruction. When the entire **IAB** microroutine has been interpreted, the mapping is again used to determine whether the IABSD postcondition indeed holds. During symbolic execution, certain *static* deductions may need to be performed. To perform a static deduction, one must prove that a state S_2 at time t was a consequence of another state S_1 at time t, with no intervening state transition. We determined that the **IAB** instruction was correctly implemented by the C/30 Microcode.

For C/30 instructions that are more complicated than the above example, the corresponding state deltas are also more complicated, and their proofs are more difficult. For instance, the C/30 shift instructions, which were implemented by iterative microcode, required inductive proofs. Certain C/30 instructions whose operation was contingent upon the current state of the machine required proof by cases. In addition, most proofs and their subproofs required static deductions.

7.3 C/30 Proof Summary

In all, 89 of the 128 C/30 instructions were proved to be correctly implemented by the C/30 Microcode. For the reasons stated in Section 5, the ten I/O, maintenance, and diagnostic instructions were not even considered. For lack of time, the verification of five multiprocessing instructions (**NMFS**, **DPR**, **SPR**, **GPR**, and **ENB**) was never completed. Minor microcode errors appeared in the microcode for 17 instructions; however, these errors did not affect the normal operation of the C/30. The microcode for five instructions was incorrect, and could result in fatal errors; an additional two instructions had microcode of dubious correctness. The erroneously microcoded C/30 instructions are the topic of the next section.

7.4 C/30 Microcode Errors

Two classes of microcode errors were discovered during the course of developing the C/30 Proof. These two classes consist of the microcode errors associated with *crash* situations and the microcode errors that lead to fatal errors.

In the MBB, the system crashes when an unrecoverable error is detected during microcode execution; a numeric *crash code* is computed before the crash occurs. Such crashes cause the

MBB to revert to a crash state under which an MBB system programmer may perform debugging operations. Most of the errors in the C/30 Microcode were associated with these crash situations. In some cases, the microcode would crash after detecting such an error, but would incorrectly set the crash code. In other cases, the microcode would not crash where a crash situation was documented; these cases may have occurred because the documentation was overly restrictive in defining errors, since in many of these situations crashing was not intuitively necessary. The C/30 instructions emulated by microcode containing crash-related errors are described as follows:

RETN, SRETN, IRETN, PUSHA, POPA, JMP, JST, PUSH, CALL, and POP all set the error code to the wrong value in the event of error. The error code values for "illegal stack pointer" and "jump to location zero" were swapped.

APR, PCB, TPR, ENB, MME, INH, and MMD did not cause a microcode crash if the MBB was not in multiprocessing mode when the instruction was executed. In addition, **APR** did not cause a crash if the process being activated was not in the idle state.

The C/30 instructions emulated by microcode containing fatal errors are described as follows:

SRC, SZC, SSC, and ACA were incorrect because of a timing error in the microcode. The parity computation for these instructions took one more microinstruction execution cycle than had originally been anticipated by the MBB microprogrammer(s).

SZO was assigned the wrong dispatch (microroutine) location by the microcode, off by one. Executing this instruction caused an "illegal instruction" trap.

LRS dispatched to one of four microprogram locations, each of which should have contained the address of the LRS microroutine, but instead contained the value zero. No dispatch memory location contained the real address of the LRS microroutine.

There were two problematic C/30 instructions, **MEMHI** and **CALL**, whose microcode could not be verified correct, but whose execution would not result in errors that could be considered fatal.

First, the **MEMHI** instruction should have assigned the highest allowable main-memory address to a C/30 register. However, the C/30 Microcode assigned the value 32K, even though the size of the C/30 main memory is 64K. Note that this anomaly is not to be considered a fatal error, as BBN advised us that the MBB microcode boot sequence patched the C/30 Microcode to correct this problem in the machine we verified.

Second, the **CALL** instruction, after pushing a return address onto the C/30's built-in stack, causes the program to branch to some memory location. Consequently, the next instruction executed would not necessarily be the instruction invoked by the call, because pushing a return address onto the stack could overwrite this memory location (i.e. the stack top location could overlap the memory location addressed by the **CALL** instruction). Note that this anomaly is also not to be considered a fatal error, as the proper management of the stack is the responsibility of the C/30 programmer.

All the fatal microcode errors were discussed with BBN, and were identified as being actual errors in the version of the microcode being verified. Because of the three-year time lag between the use of this microcode in the field and its verification, we were not surprised to learn that all of the fatal microcode errors had been reported to BBN and had been corrected in newer versions of the microcode.

8 Conclusions

The major successes of the C/30 Microcode Verification Project were the formal verification of the correctness of approximately 1000 lines of C/30 Microcode (proving the correctness of the microcode that implements a majority of the C/30's instructions, and identifying numerous microcode errors), as well as a demonstration of SDVS's ability to tackle large-scale verification efforts.

With respect to the design of the computers and microprograms at issue in this study, the correctness of hardware and software could never be certified solely by testing. However, if tests of such descriptions or programs are coupled with formal verification in CAD/CAM or CASE environments, then the physical implementation of computers and their software will have a much higher probability of being correct. In particular, coupling the testing and debugging process with microcode verification should result in microcode whose reliability is significantly increased, with greatly reduced maintenance costs and a need for fewer microcode updates.

Other issues of concern involve aspects of SDVS and ISPS. The ISPS specifications of the MBB and the C/30 took more than two years to write and required additional time to debug. More than one year was required to develop the C/30 Proof and theorems, which consists of approximately 600 pages of text. The actual computer time required to check the correctness of the C/30 Proof on a Symbolics 3640 was approximately 85 hours. Of course, the computer that verified the C/30 Proof is now at least four years old, and we have observed current computers capable of an eight-fold increase in the execution speed of SDVS. Further reductions in the time required for verification can be achieved by simply having in hand the hardware and software specifications of a given design.

All of these times could be reduced, however, because ideally hardware and software specifications would provide the basis for computer and software design, and the verification process could be folded into the design and implementation process.

The C/30 Microcode Verification Project was completed in 1986. Since then, many improvements have been made to SDVS. Given the proper data, SDVS is now capable of automatically constructing the statement of implementation correctness. In addition, SDVS has a new translator for a larger subset of ISPS. A formal denotational semantics [24] for the new translator has been specified in the internal language of DENOTE [25], which automatically generates a Common Lisp [26] implementation of the translator. Because of the inadequacies of ISPS as an HDL, VHDL (VHSIC hardware description language) is now being considered by the developers of SDVS for the verification of hardware designs [27, 28]. In addition, as described in [13], we have added Ada verification capabilities to SDVS, and are continuing to incorporate larger subsets of the language.

Acknowledgments

The author thanks his colleagues Steve Crocker, Leo Marcus, Sue Landauer, Tim Redmond, and Eve Cohen for contributing to this project in 1985-86; Mel Cutler for the ISPS description of the C/70 MBB; the BBN employees who provided project support; and Dave Martin, Beth Levy, and Mel Cutler for providing some of the material included in this report.

References

[1] Bolt, Beranek, and Newman, Inc., "C/30 Native Mode Firmware System, Programmer's Reference Manual," Tech. Rep. 5000, Bolt, Beranek, and Newman, Inc., Nov. 1983. This document contains BBN proprietary information and is not available to the public.

[2] A. Lake *et al.*, "Flexible processor extends design options," *Computer Design*, pp. 181–186, Nov. 1981.

[3] P. Herman, M. Kraley, and R. Weissler, "MBB Microprogrammer's Handbook," Tech. Rep. 4268, Bolt, Beranek, and Newman, Inc., Aug. 1980. This document contains BBN proprietary information and is not available to the public.

[4] L. Marcus, "SDVS 8 Users' Manual," Tech. Rep. ATR-89(4778)-4, The Aerospace Corporation, Sept. 1989.

[5] L. Marcus, T. Redmond, and S. Shelah, "Completeness of State Deltas," Tech. Rep. ATR-86(8454)-2, The Aerospace Corporation, 1986.

[6] M. J. C. Gordon, "HOL—a proof generating system for higher order logic," in *VLSI Specification, Verification, and Synthesis* (G. Birtwistle and P. Subrahmanyam, eds.), Kluwer, 1987.

[7] W. J. Cullyer, "Implementing safety-critical systems: The VIPER microprocessor," in *VLSI Specification, Verification, and Synthesis* (G. Birtwistle and P. Subrahmanyam, eds.), Kluwer, 1987.

[8] A. Cohn, "A proof of correctness of the VIPER microprocessor: The first level," in *VLSI Specification, Verification, and Synthesis* (G. Birtwistle and P. Subrahmanyam, eds.), Kluwer, 1987.

[9] R. S. Boyer and J. S. Moore, "A theorem-prover for recursive functions; a user's manual," Tech. Rep. CSL-91, SRI International, 1979.

[10] J. Warren A. Hunt, "Fm8501: A verified microprocessor," Tech. Rep. Technical Report 47, Institute for Computing Science, The University of Texas at Austin, Feb. 1986.

[11] L. Marcus, S. D. Crocker, and J. R. Landauer, "SDVS: A system for verifying microcode correctness," in *17th Microprogramming Workshop*, pp. 246–255, IEEE, Oct. 1984.

[12] L. Marcus, "SDVS 5 Users' Manual," Tech. Rep. TR-0086(6778)-2, The Aerospace Corporation, 1986.

[13] D. F. Martin and J. V. Cook, "Adding Ada program verification capability to the State Delta Verification System (SDVS)," in *Proceedings of the 11th National Computer Security Conference*, (Baltimore, Md.), National Bureau of Standards/National Computer Security Center, Oct. 1988.

[14] M. M. Cutler, "Verifying implementation correctness using the State Delta Verification System (SDVS)," in *Proceedings of the 11th National Computer Security Conference*, (Baltimore, Md.), National Bureau of Standards/National Computer Security Center, Oct.17–20 1988.

[15] M. R. Barbacci, G. E. Barnes, R. G. Cattell, and D. P. Siewiorek, "The ISPS Computer Description Language," Tech. Rep. CMU-CS-79-137, Carnegie-Mellon University, Computer Science Department, Aug. 1979.

[16] E. Cohen and J. Landauer, "Specification Problems Encountered during the Proof of the C/30 Microcode," Tech. Rep. ATR-86(6778)-2, The Aerospace Corporation, 1986. This document may contain BBN proprietary information.

[17] B. H. Levy, "Inadequacies of ISPS as a Specification Language for Microcode Verification," Tech. Rep. ATR-86A(2778)-1, The Aerospace Corporation, 1987.

[18] S. D. Crocker and M. M. Cutler, "A Formal Description of the Microarchitecture of the C/70 Computer," Tech. Rep. ATM 82(2920-03)-1, The Aerospace Corporation, Mar. 1982. This document contains BBN proprietary information and is not available to the public.

[19] J. V. Cook, S. D. Crocker, and M. M. Cutler, "A Formal Description of the Microprogrammable Building Block Configured for the C/30 Computer," Aerospace Technical Report ATR-86(6771)-1, The Aerospace Corporation, 1986. This document contains BBN proprietary information and is not available to the public.

[20] J. V. Cook, "A Formal Description of the C/30 Virtual Computer," Aerospace Technical Report ATR-86(6771)-2, The Aerospace Corporation, 1986. This document may contain BBN proprietary information.

[21] J. V. Cook, "C/30 Proof," Tech. Rep. ATR-86(6771)-4, The Aerospace Corporation, Sept. 1986. This document contains BBN proprietary information and is not available to the public.

[22] J. V. Cook, "Final Report for the C/30 Microcode Verification Project," Aerospace Technical Report ATR-86(6771)-3, The Aerospace Corporation, Sept. 1986. This document may contain BBN proprietary information.

[23] J. V. Cook, "Proof Strategy for the Verification of the C/30 Microcode," Aerospace Technical Report ATR-86(6778)-1, The Aerospace Corporation, Sept. 1986. This document may contain BBN proprietary information.

[24] T. Aiken, "A Revised Formal Description of the Incremental Translation of ISPS into State Deltas in the State Delta Verification System (SDVS)," Tech. Rep. ATR-90(5778)-1, The Aerospace Corporation, 1990.

[25] J. V. Cook, "The Language for DENOTE (Denotational Semantics Translator Environment)," Technical Report TR-0090(5920-07)-2, The Aerospace Corporation, 1989.

[26] Guy L. Steele Jr., *Common LISP: The Language.* Digital Press, 1984.

[27] B. H. Levy and I. V. Filippenko, "A Preliminary SDVS Semantics of a VHDL Subset," Technical Report ATR-88(3778)-7, The Aerospace Corporation, Aug. 1988.

[28] T. Aiken, I. Filippenko, B. Levy, and D. Martin, "A Formal Description of the Incremental Translation of Core VHDL into State Deltas in the State Delta Verification System (SDVS)," Tech. Rep. ATR-89(4778)-9, The Aerospace Corporation, 1989.

Data Categorization and Labeling

PANEL SESSION OVERVIEW

Dr. Dennis K. Branstad, Chairman
Senior Computer Science Fellow
National Institute of Standards and Technology

The purpose of a security label is to provide information for an intended recipient of a document or data regarding the desired protection to be provided. A label can explicitly state what protection to provide, e.g., DO NOT FOLD, MUTILATE OR DESTROY. A label can implicitly state what protection to provide, e.g., SECRET. The explicit protection requirements for implicitly labeled data are contained in separate legislation, policy, directives and instructions. This session outlines several categories of information requiring protection and discusses security labels for the categories that would implicitly include the protection required. Security labels that could be used for routing purposes in an Internet is presented.

I. Security Labels: Scope and Purpose

A security label is a short-hand notation denoting either a category of information to be protected or the protection to be provided. IBM PROPRIETARY and U.S. SECRET are examples of the former and DO NOT COPY is an example of the latter. The Internet Protocol Security Option (IPSO) Label is an example of an electronic label that can be attached to every Network Layer packet of data that denotes its classification and certain other relevant security information. This label can be used by network intermediate systems (e.g., routers, gateways) to determine which route a packet will take to its destination.

A security label should contain enough information, either explicitly or implicitly, for any potential, intended receiver to know how to protect the received data. Standards are required for security labels so that this protection can be universal, or nearly so. The standards either need to specify the format and contents of a label completely or provide an extensible format so that the contents can vary widely within certain ranges. The semantics of a label can then be obtained from some source (e.g., a registration authority) so that the proper protection is provided.

The results of Federal standards development should include an extensible security label format that would satisfy a wide range of protection requirements. Protection must include confidentiality and integrity and in some circumstances would include availability and timeliness. A label itself requires integrity and availability protection but should not require (at least preferably) confidentiality protection. In addition, a wide range of commercial security requirements should be considered when defining the label format.

II. Panel Presentations

This session includes three presentations on information categorization and labeling. The first presentation will give a broad overview of information protection requirements and various security categories into which information may be placed. The second presentation will include considerations of security labels in the Open Systems Interconnection (OSI) communications model. The final presentation will cover security labeling in unclassified networks and dwell upon the results of a NIST hosted workshop on security labels held in May, 1990.

INFORMATION CATEGORIZATION AND PROTECTION

The need to understand the value of information.

Warren Schmitt
Sears Technology Services, Inc.

Information can be represented in many forms. It can originate from the spoken word, it can be written, or it can be digitized and transformed into electronic form. In some situations, the same information may have different meaning to different people. Hard-copy information seems to engender a different reaction from "invisible" information that is transmitted and stored electronically. Sometimes the old expression, "out of sight, out of mind" seems to take precedence with electronic information printed on paper would be well protected. No sooner having said that, then someone would give an example where there is substantially greater protection given information on a computer than when the information produced in a report.

Maybe these differing perceptions are some reasons why it is difficult to place a value on information. And why in some communities, like the intelligence community and the Department of Defense, they take great pains to protect information from disclosure, while others treat information with a rather cavalier attitude, and pay little attention to protecting it. And still others take the position that all information should be free and available to anyone who wishes to have access.

These widely differing points of view may explain in part why there hasn't been any substantive effort to analyze both the vulnerabilities and the related protective measures that are associated with information and to understand to what degree the three major risks, destruction, modification, and disclosure, may impact the asset called information. A categorization process such as this would identify how valuable or susceptible the information is, and provide bench marks for its protection.

In the commercial sector and the civilian government agencies, the value of information is not substantially different today than it was forty years ago. The confidentiality surrounding salary information, for example, is about the same today as then. Research findings that would lead to the envelopment of a new product were as valued then as now. The integrity of financial records still demand great care and diligence and the ability to recover information from a damaging event still remains a significant management concern.

However, many things about information have changed in the last forty years. Most notably, how we gather, manipulate, distribute, and store information. And most of these changes center around the subscription fees etc., the information does not command the same degree of confidentiality as would seismic information about a future drilling site, or the plans for corporate mergers or acquisitions.

The categorization for some risks can be done at relatively high levels. For example, an entire application may be categorized as high/low as it pertains to availability (disaster recovery) while a more detailed break-down, to isolate a program or process, may be necessary to identify the degree of concern for the integrity or confidentiality of the information.

The guardian, or the organizational entity that is responsible or the accuracy and integrity of the information, is the best source to categorize the information for each of the major risks. He may need some help from his application development staffs to better understand the applications and programs.

The flip side of the categorization process is the identification of the controls that would best protect the information from the agreed upon risks. By and large this has been left up to the application designer with some limited input from the internal auditor. The control identification process needs to be greatly strengthened to include the Guardian, the application designer, the custodian (usually data processing), the user, and internal auditing. Many of the controls can be pre-approved for use in all applications, while other controls will have to be selected based on the individual application.

As the categorization process progresses, a data base should be established. This data base would identify the information, the categorization assigned, the authority who established the categorization (usually the Guardian), and the date it was approved.This data base should be periodically reviewed to insure its accuracy, usually on an annual basis. If there is a question as to whether the categorization is appropriate, the data base will be the source to identify the author. Additionally it would also be used by the application programmer to identify the categorization and be able to understand the level of controls that are appropriate for the application.

The establishment of categories as they relate to information is often referred to as labels. The labels could become an integral part of the information, particularly when new applications are designed, to ensure that the proper controls are established, or they could reside in a repository. Establishing labels in this manner would help ensure that, once the information had been categorized and the appropriate controls had been established, this information could be carried forward as the applications are revised or rewritten.

Information Technology is a very complex discipline and as this technology becomes more complex we must establish a systematic process whereby we can analyze the risks to which information is exposed and identify the appropriate controls. Unless we approach Information Security differently than we've done in the last 15 years, we're destined to manage information the same way during the next 15 years. In the long term we will be judged by how well we managed our information rather than on how uniquely it was processed. Concentrating on the value of information may be the key.

innovations embodied in Information Technology.

By contrast with today, at the half-way point in this century, the person responsible for the accuracy and the integrity of the information was the person who most often had the information in his custody and intimately knew its value and associated risks. This enabled him, in many cases, to personally apply the controls he deemed appropriate to protect the information. The information technology revolution has, however, dramatically changed the way we must manage information. The person today who is responsible for the accuracy and integrity of the information (whom we shall call Guardian) frequently does not have the information in his custody. He must rely on data processing and networking personnel who have the physical custody and control of the information, the input, the transmission, and the processing. These persons should be thought of as custodians of the information with highly skilled functions to perform. Other than in very general terms, these technicians are not aware of the value of the information. If the value of the information and appropriate controls are not stipulated by the Guardian, it is not reasonable to expect that all the necessary controls will be in place.

Twenty years ago, before networking and distributed processing become major implementation strategies, the data processing functions, by default, usually assumed the responsibility for protecting information. And by and large, because of the centralized nature of the processing, they did a rather effective job. Today, the user has become accustomed to relying on his information technology support staffs to design his applications and provide processing and telecommunications capabilities. Each of these functions has developed into highly technical functions whereby we have become specialist in our own domains. And as a result, we have become insulated from the true value of the information as it relates to the enterprise.

Tomorrow, as the control of applications is vested in the end-user, we will see even greater changes in the field of Information Technology. If we are to place ourselves in a position to properly manage our information assets in this rapidly changing environment, we need to implement a well-organized, systematic approach to the identification of the risk factors associated with information and the generally accepted controls that may be employed to protect the information.

One solution would be to categorize the information we maintain on computer systems in terms of the information's susceptibility to each of the major risks mentioned above i.e., destruction, modification, and disclosure. For the sake of this discussion I have reserved the normal order in which these risks are usually listed in order to place emphasis on the fact that disclosure is not the major concern of the commercial sector. From the business community's perspective, each of the risks can be generally considered as equally important.

An important aspect to remember is that not all information is equally susceptible to each of these three major risks. For example, airline reservation information may be ranked very high from the standpoint of integrity and availability. The providers of this information would naturally be very concerned with the correctness of the information and would want the information to be readily available in both a printed and an on-line format. Although the service provider has strong concerns about disclosure from the stand point of authorized users and the related

Security Labels in Open Systems Interconnection

Russell Housley
Xerox Special Information Systems
McLean, Virginia

INTRODUCTION

This paper presents a security labeling framework for open systems interconnection (OSI)[1]. The framework is intended to help protocol designers determine what, if any, security labeling should be supported by their protocol. The framework should also help network architects determine whether or not a particular collection of protocols fulfill all of their security labeling requirements.

SECURITY LABELS

Data security is the measures taken to protect data from accidental, unauthorized, intentional, or malicious modification, destruction, or disclosure. Data security is also the condition that results from the establishment and maintenance of protective measures[2]. Given this two-pronged definition for data security, security labeling as one mechanism which provides data security will be examined. In general, security labeling by itself can not provide sufficient data security; it must be complemented by other security mechanisms.

In OSI, security labels tell the protocol processing how to handle the data communicated between two open systems. That is, the security label indicates what measures need to be taken to preserve the condition of security. "Handle" denotes the activities performed on data such as collecting, processing, transferring, storing, retrieving, sorting, transmitting, disseminating, and controlling[3].

The definition of data security includes protection from modification and destruction. That is, protection from writing and deleting. These protections are the data integrity service defined in the OSI Security Architecture[4].

Biba[5] has defined a data integrity model which includes security labels. The Biba model specifies controls for writing and deleting in order to preserve data integrity. The model also specifies control for reading to ensure that data is not copied to a container where integrity can not be guaranteed.

Our definition of data security also includes the protection from disclosure. That is, protection from reading. This protection is the data confidentiality service defined in the OSI Security Architecture[4].

Bell and LaPadula[6] defined a data confidentiality model which includes sensitivity labels. The Bell and LaPadula model specifies controls for reading in order to preserve data confidentiality. The model also specifies control for writing to ensure that data is not copied to a container where confidentiality can not be guaranteed.

Notice that in both the Biba model and the Bell and LaPadula model, the security label is an attribute of the data. In general, the security label associated with the data will remain constant. Exceptions will be discussed later in the paper, but any relabeling is always the result of some network entity handling the data.

INTEGRITY LABELS

Integrity labels (like those defined in the Biba model) support rule-based access control (RBAC) policies. The integrity label tells the degree of confidence that may be placed in the data and also tells which measures the data requires for protection from modification and destruction.

As data moves through the network, it may be relabeled with a lower integrity label as a result of being handled by an entity with an integrity label lower than the data's integrity label. When this happens, the data is relabeled with the label of the entity. As data moves through the network, it may never be relabeled with a higher integrity label.

One of the rules in the access control policy might prohibit this relabeling. In this case, data may only be handled by entities which have the same or a higher integrity label than the data.

Each of the open systems on a network must include RBAC policies and the protocol suite must transfer the integrity label with the data if the confidence of the data is to be maintained throughout the network. Each of the open systems on a network may have it's own internal representation for a integrity label, but the protocols must provide common syntax and semantics for the transfer of the integrity label (as well as the data itself).

To date, no protocols have been standardized which include integrity labels in the protocol control information.

SENSITIVITY LABELS

Sensitivity labels (like those defined in the Bell and LaPadula model) support rule-based access control (RBAC) policies. The sensitivity label tells the amount of damage that will result from the disclosure of the data and also tells which measures the data requires for protection from disclosure.

As data moves through the network, it may be relabeled with a higher sensitivity label as a result of being handled by an entity with a sensitivity label higher than the data's sensitivity label. When this happens, the data is relabeled with the sensitivity label of the entity. As data moves through the network, it may never be relabeled with a lower sensitivity label.

One of the rules in the access control policy might prohibit this relabeling. In this case, data may only be handled by entities which have the same sensitivity label that the data. (Entities with lower sensitivity labels may not handle the data; this would be disclosure. Entities with higher sensitivity labels may not handles the data either; this would cause the data to be upgraded.)

Each of the open systems on a network must include RBAC policies and the protocol suite must transfer the sensitivity label with the data if the protection from disclosure is to be maintained throughout the network. Each of the open systems on a network may have it's own internal representation for a sensitivity label, but the protocols must provide common syntax and semantics for the transfer of the sensitivity label (as well as the data itself).

Sensitivity labels, like the ones provided by the IP Security Option (IPSO)[6], have been used in networks for years.

SECURITY LABEL REQUIREMENTS

OSI defines two major types of systems: end systems and intermediate systems[1]. These terms should be familiar to the reader. For this discussion, however, the traditional definition of intermediate system will be broadened to include routers, packet switches, and bridges. End systems and intermediate systems have different security label requirements.

END SYSTEM SECURITY LABEL REQUIREMENTS

When two end systems communicate, a common security label syntax and semantics are needed. The security label, as an attribute of the data, indicates what measures need to be taken to preserve the condition of security. The security label must communicate all of the integrity and confidentiality handling requirements. These handling requirements can become very complex.

Some operating systems label the data they process. These security labels are not part of the data, rather they are attributes of the data. Some database management systems (DBMSs) perform similar labeling. The format of these security labels is a local matter, but they are usually in a format different than the one used by the network protocols.

Trusted operating systems which implement RBAC policies require security labels on the data they import[8,9]. These security labels permit the Trusted Computing Base (TCB) in the end system to perform trusted demultiplexing. That is, the network traffic is relayed from the TCB to a process only if the process has sufficient authorization for the data. In most cases, the TCB must first translate the network security label into the local syntax before it can make the access control decision.

INTERMEDIATE SYSTEM SECURITY LABEL REQUIREMENTS

This is a discussion of "user" data security labels within the intermediate system. The labeling requirements associated with intermediate system-to-end system (IS-ES) traffic, intermediate system-to-intermediate system (IS-IS) traffic, and intermediate system-to-network management (IS-NM) traffic are not included in this discussion.

Intermediate systems make routing choices or discard traffic based on the security label. The security label used by the intermediate system should contain only enough information to make the routing/discard decision and may be a subset of the security label used by the end system. For example, handling restrictions (like WNINTEL) are unlikely to effect routing decisions, but they may effect processing done within the end system.

In most networks, very few intermediate systems actually make access control decisions. For performance reasons, only those intermediate systems which do make access control decisions should be burdened with parsing the security label. That is, information hiding principles apply.

Intermediate systems do not usually translate the network security labels to a local format. They use them "as is" to make their routing/discard decisions. However, when two classification authorities share a network by bilateral agreement, the intermediate systems may be required to perform label translation. For example, assume that there are two Department of Energy (DOE) accredited subnets attached to a Department of Defense (DOD) wide area network (WAN). Routers between a DOE subnet and the DOD WAN must translate DOE labels to DOD labels so that the routers within the DOD WAN can make appropriate routing decisions.

APPROACHES TO LABELING

There are several tradeoffs to be made when determining how a particular network will perform security labeling. Explicit or implicit labels can be used. Also, security labels can either be connectionless or connection-oriented.

EXPLICIT VS. IMPLICIT SECURITY LABELS

Explicit security labels are actual bits in the protocol control information (PCI). The IP Security Option (IPSO) is an example of an explicit security label[7]. Explicit labels may be either connectionless or connection-oriented.

Implicit security labels are not actual bits in the PCI, rather some attribute is used to determine the security label. For example, the choice of cryptographic key in the SP4 protocol[10,11] can determine the security label. Implicit labels may be either connectionless or connection-oriented.

CONNECTIONLESS VS. CONNECTION-ORIENTED SECURITY LABELS

When connectionless security labels are used, the security label appears in every protocol data unit (PDU). All protocols have limits on the size of their PCI, and the explicit security label may not exceed this size limit. It can not use the entire PCI space either; the protocol has other fields that must be transferred as well. This size limitation may prohibit explicit connectionless security labels from meeting the requirements of end systems. However, the requirements of intermediate systems are fully satisfied by explicit connectionless security labels. The IP Security Option (IPSO)[7] is an example of connectionless labeling.

Connection-oriented security labels are attributes of virtual circuits, connections, and associations (for simplicity, all of these are subsequently referred to as connections). The security label is defined at connection establishment, and all data transferred over that connection inherits that security label. This approach is more compatible with end system requirements than intermediate system requirements. One noteworthy exception is X.25 packets switches; these intermediate systems could associate connection-oriented labels with each virtual circuit. One example of connection-oriented security labels involves two protocols: the SDNS Key Management Protocol (KMP)[12,13,14] can be used to associate security labels with each of the transport connections protected by the SP4 protocol[10,11] (using SP4C).

Connectionless security labels may be used in conjunction with connectionless or connection-oriented data transfer protocols. However, connection-oriented security labels may only be used in conjunction with connection-oriented data transfer protocols.

LABELING WITHIN THE OSI REFERENCE MODEL

Each of the seven OSI layers will be examined with respect to security labels. Figure 1 illustrates the well known reference model. Layer 1, the physical layer, will be examined first. Then, each successively higher layer will be examined.

LAYER 1, THE PHYSICAL LAYER

Explicit security labels are not possible in the Physical Layer. The Physical Layer does not include any protocol control information (PCI), so there is no place to include the bits which represent the label.

Implicit security labels are possible in the Physical Layer. For example, all of the data that comes in through a particular physical plug could inherit one security label. Most physical connections are connectionless (they support only bit-at-a-time or byte-at-a-time operations), so these implicit security labels are connectionless.

Implicit security labels in the Physical Layer may be used to meet the requirements of either end systems or intermediate systems so long as the physical connection is single level. That is, only one security label is associated with all of the data received or transmitted through the physical connection.

LAYER 2, THE DATA LINK LAYER

Explicit security labels are possible in the Data Link Layer. In fact, the IEEE 802.2 Working Group is currently working on an optional security label standard for the Logical Link Control (LLC) protocol (a.k.a. IEEE 802.2)[15]. These labels will optionally appear in each LLC frame. These are obviously connectionless security labels.

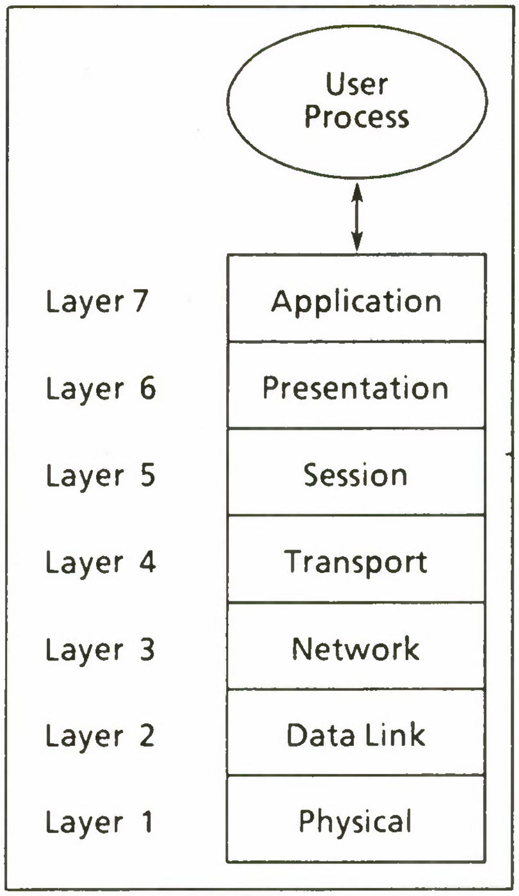


Figure 1. OSI Reference Model.

Explicit connection-oriented security labels are also possible in the Data Link Layer. One could imagine a security label standard which worked with LLC Type II.

Of course, implicit security labels are also possible in the Data Link Layer. These implicit labels could be either connectionless or connection-oriented. One attribute that might be used in IEEE 802.3 (CSMA/CD)[16] to determine the implicit security label is the source address of the frame.

Security labels in the Data Link Layer may be used to meet the requirements of end systems and intermediate systems. Explicit security labels in this layer tend to be small, so end systems with requirements for large security labels should use a higher protocol layer. However, label-based routing decisions made by bridges are best supported in this layer.

LAYER 3, THE NETWORK LAYER

Explicit security labels are possible in the Network Layer. In fact, the IP Security Option (IPSO) has been used for many years. These labels optionally appear in each IP datagram. IPSO labels are obviously connectionless security labels.

Explicit connection-oriented security labels are also possible in the Network Layer. One could easily imagine a security label standard for X.25[17].

Of course, implicit security labels are also possible in the Network Layer. These implicit labels could be either connectionless or connection-oriented. One attribute that might be used to determine the implicit security label is the X.25 virtual circuit.

Security labels in the Network Layer may be used to meet the requirements of end systems and intermediate systems. Explicit security labels in this layer tend to be small, so end systems with requirements for large security labels should use a higher protocol layer. Alternatively, the Network Layer (especially the the Subnetwork Independent Convergence Protocol (SNICP)) is an excellent place to carry a security label to support trusted demultiplexing because many implementations demultiplex from an system-wide daemon to a user process after network layer processing. The SNICP is end-to-end, yet it is low enough in the protocol stack to aid trusted demultiplexing.

Label-based routing decisions made by routers and packet switches are best supported in the Network Layer. Routers can also add security labels at subnetwork boundaries. However, placement of these security labels must be done carefully to ensure that the addition of the security label does not degrade overall network performance by forcing routers that do not make label-based routing decisions to parse the security label.

LAYER 4, THE TRANSPORT LAYER

Explicit security labels are possible in the Transport Layer. In fact, the SP4 protocol[10,11] includes them. These security labels can be either connectionless (using SP4E) or connection-oriented (using SP4C). SP4 is an addendum to the TP[18] and CLTP[19] protocols.

Implicit security labels are also possible in the Transport Layer. These implicit labels could be either connectionless or connection-oriented. One attribute that might be used to determine the implicit label in the SP4 protocol (when explicit labels are not used as discussed above) is the choice of cryptographic key.

Security labels in the Transport Layer may be used to meet the requirements of end systems. The Transport Layer, being end-to-end can not be used to meet the requirements of intermediate systems. Connection-oriented explicit security labels in this layer are especially good for meeting end system requirements where large labels are required. The label is only transmitted at connection establishment, so overhead is kept to a minimum. Yet, in many implementations the Transport Layer is low enough in the protocol stack to aid trusted demultiplexing.

LAYER 5, THE SESSION LAYER

Explicit security labels are possible in the Session Layer. Session Layer security labels could be either connectionless or connection-oriented. However, it is unlikely that a standard will ever be developed for such labels because the OSI Security Architecture[4] does not allocate any security services to the Session Layer.

Implicit security labels are also possible in the Session Layer. These implicit labels could be either connectionless or connection-oriented. Again, the ISO Security Architecture makes this layer an unlikely choice for security labeling.

Security labels in the Session Layer may be used to meet the requirements of end systems, but the Session Layer is too high in the protocol stack to be used to meet the requirements of intermediate systems. The Session Layer is also too high in the protocol stack to support trusted demultiplexing.

LAYER 6, THE PRESENTATION LAYER

Explicit security labels are possible in the Presentation Layer. The presentation syntax may include a security label. This approach naturally performs translation to the local label format. This approach supports connectionless and connection-oriented security labeling.

Implicit security labels are also possible in the Presentation Layer. These implicit security labels could be either connectionless or connection-oriented.

Security labels in the Presentation Layer may be used to meet the requirements of end systems, but the Presentation Layer is too high in the protocol stack to be used to meet the requirements of intermediate systems. The Presentation Layer is also too high in the protocol stack to support trusted demultiplexing.

LAYER 7, THE APPLICATION LAYER

Explicit security labels are possible in the Application Layer. The CCITT message handling system includes security labels in message envelopes[20]. Other Application Layer protocols will probably include security labels in the future. These security labels could be either connectionless or connection-oriented. It is most likely that transaction processing protocols and message handling protocols will include connectionless security labels; other application protocols will most likely include connection-oriented security labels.

Application layer protocols are unique in that they can include security label information which is specific to a particular application without burdening other applications with the syntax or semantics of that security label.

Implicit security labels are also possible in the Application Layer. These implicit security labels could be either connectionless or connection-oriented. One attribute that might be used to determine the implicit label is the application title.

Security labels in the Application Layer may be used to meet the requirements of end systems, but the Application Layer is too high in the protocol stack to be used to meet the requirements of intermediate systems. The Application Layer is also too high in the protocol stack to support trusted demultiplexing.

SUMMARY

As we have seen, very few hard rules exist for security labels in OSI. Protocol designers and network architects are faced with many tradeoffs when making security label placement decisions. A few guidelines can be derived from the preceding discussion.

Security label-based routing decisions are best supported by explicit security labels in the Data Link Layer and the Network Layer. It is no surprise that when bridges are making the routing decisions, the Data Link Layer should carry the explicit security label; when routers are making the routing decisions, the Network Layer should carry the explicit security label.

When security labels are specific to a particular application, it is wise to define them in the application protocol where these security labels will not burden other applications on the network.

When trusted demultiplexing is a concern, the Network Layer (preferably the SNICP) or Transport Layer should be used to carry the explicit security label.

REFERENCES

[1] *ISO Open Systems Interconnection - Basic Reference Model* (ISO 7498). International Organization for Standardization, 1981.

[2] *Dictionary of Military and Associated Terms* (JCS Pub 1). Joint Chiefs of Staff. 1 April 1984.

[3] *Security Requirements for Automatic Data Processing (ADP) Systems* (DODD 5200.28). Department of Defense. 21 March 1988.

[4] *Information Processing Systems - Open Systems Interconnection Reference Model - Security Architecture* (ISO 7498-2). Organization for Standardization, 1988.

[5] Biba, K. J. "Integrity Considerations for Secure Computer Systems," MTR-3153, The Mitre Corporation, April 1977.

[6] Bell, D. E.; LaPadula, L. J. "Secure Computer System: Unified Exposition and Multics Interpretation," MTR-2997, The Mitre Corporation, March 1976.

[7] St. Johns, M. "Draft Revised IP Security Option," RFC 1038, Network Information Center (NIC), January 1988.

[8] *Trusted Computer System Evaluation Criteria* (DoD 5200.28-STD) National Computer Security Center, 26 December 1985.

[9] *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria*, (NCSC-TG-005, Version-1). National Computer Security Center, 31 July 1987.

[10] *Security Protocol 4 (SP4)*, SDN.401. Secure Data Network Systems (SDNS) Special Program Office, 1989.

[11] Branstad, D.;Dorman, J.; Housley, R.; Randall J. "SP4: A Transport Encapsulation Security Protocol." *AIAA/ASIS/IEEE Third Aerospace Computer Security Conference: Applying Technology to Systems*. December 1987; pp. 143-147.

[12] *Key Management Protocol: Definition of Services Provided by the Key Management Application Service Element*, SDN.902. Secure Data Network Systems (SDNS) Special Program Office, 1989.

[13] *Key Management Protocol: Specification of the Protocol for Services Provided by the Key Management Application Service Element*, SDN.903. Secure Data Network Systems (SDNS) Special Program Office, 1989.

[14] *Key Management Protocol: SDNS Traffic Key Attribute Negotiation*, SDN.906. Secure Data Network Systems (SDNS) Special Program Office, 1989.

[15] *IEEE Standards for Local Area Networks: Logical Link Control*, IEEE 802.2. The Institute of Electrical and Electronics Engineers, Inc, 1984.

[16] *IEEE Standards for Local Area Networks: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specification*, IEEE 802.3. The Institute of Electrical and Electronics Engineers, Inc, 1985.

[17] *Recommendation X.25, Interface Between Data Terminal Equipment (DTE) and Data Circuit Terminating Equipment (DCE) for Terminals Operating in the Packet Mode on Public Data Networks*. Consultative Committee, International Telephone and Telegraph (CCITT), 1984.

[18] *Information Processing Systems - Open Systems Interconnection - Connection oriented transport protocol specification* (ISO 8073). Organization for Standardization, 1985. [Also ISO 8208]

[19] *Information Processing Systems - Open Systems Interconnection - Protocol for providing the connectionless-mode transport service* (ISO 8602). Organization for Standardization, 1986.

[20] *Recommendation X.411, Message Handling Systems: Message Transfer System: Abstract Service Definition and Procedures*. Consultative Committee, International Telephone and Telegraph (CCITT), 1988. [Also ISO 8883-1]

ACKNOWLEDGEMENTS

The members of the Privacy and Security Research Group and the attendees of the invitational Security Labels Workshop (hosted by the National Institute of Standards and Technology) helped me organize my thoughts on this subject. The ideas of these professionals are scattered throughout the paper.

Executive Summary

Security Labeling in Unclassified Networks

Noel A. Nazario
Protocol Security Group
National Institute of Standards and Technology

Introduction

As computer networks become widespread, government agencies and commercial operations are expressing a need to safeguard unclassified information that is sensitive to their operations. Security labels are used to provide data handling instructions to the network protocol processing [4]. These handling indications reflect the security policy of the organization that owns the data. Existent systems that use labeling reflect the security policies of the Department of Defense which do not address the needs of the unclassified community. A different set of requirements must be considered in addressing the needs of this community. In order to stimulate the development of off-the-shelf products that provide appropriate protection, it is necessary to devise an approach to security labels that can be acceptable to both the classified and unclassified communities.

The National Institute of Standards and Technology has taken an active role in the development of the U.S. Government Open System Interconnection Profile (GOSIP) [8]. This profile attempts to define a set of requirements, which include security, that will be used by the U.S. Government in the procurement of computer communications equipment. The development of GOSIP is also being watched closely by non-government users of secure computer communications. NIST works in partnership with the National Computer Security Center drawing upon its technology and products to provide solutions to the computer security needs of the government unclassified and commercial communities. Security labels was identified by NIST as one of the areas that need prompt attention in the development of a unified approach to Open Systems Interconnection (OSI) security.

On May 30 and 31, 1990, NIST hosted an invitational workshop to address security labels for open systems. This workshop provided a forum for users to express their needs as well as to receive the technical contributions of experts in network security. After publishing the proceedings of this workshop [6] NIST staff will draft text on security labeling for Chapter 6 of GOSIP.

Background Information

Any standardization activity for network security labels has to take into consideration previous work in this area. The labeling approach most widely used is the Internet Protocol Security Option (IPSO).

Initially the IPSO was described in the original TCP/IP protocol specifications in the DDN Protocol Handbook [2]. It was probably never implemented until Captain Michael St. Johns, USAF, drafted RFC 1038 [7] in 1988. The Arpanet Request for Comment (RFC) 1038 defines the IP Basic and Extended security option fields and indicates how to use them in enhancing network security.

Other related efforts, such as the Commercial IPSO (CIPSO) [5], have been undertaken recently in attempt to expand the applicability of the original specification.

Currently, the GOSIP document defines a security option for the Connectionless Network Protocol (CLNP) which is almost identical to the Revised IPSO.

Two specifications for secure protocols for OSI, SP3 and SP4 [3], are currently been presented by NIST to the American National Standards Institute (ANSI) and the International Standardization Organization (ISO). These specifications were created under the Secure Data Network System (SDNS) program and released to the public domain with NIST as the custodian. They describe two secure protocols for the Network and Transport Layers of the OSI architecture and include fields for security labels that are not well defined but are nevertheless available. It seems likely that SP3 and SP4 labels will be accepted in some form as international standards and eventually included in GOSIP. It has been suggested that the final format adopted for security labels at both layer 3 and layer 4 should be the same.

Classified vs. Unclassified Requirements

When looking at unclassified network security we find that one of the main problems is the introduction of multiple security domains. A security domain is a collection of interconnected systems that operate under a common security policy. This means that the definitions of clearances and sensitivity categories may be different and, in some instances, non-transferable across domains. User organizations can define security policies appropriate to their operations that may not necessarily apply to any other organization. In the classified sector, for instance, there are four basic classification levels: unclassified, confidential, secret, and top-secret. These basic classifications are complemented with categories, or compartments, and markings. The definition and usage of these attributes are given in a well defined security policy oriented towards the needs of the classified community.

Even though some of the differences are fundamental, they are not necessarily unsolvable. A good number of similarities do exist. Both communities need to make rule based access control (RBAC) decisions based on the information carried by the label. There is a common requirement to indicate what measures are needed to protect information against unauthorized disclosure. Also common is the need to indicate measures against unauthorized modification and the confidence that may be placed on the information.

Communications within the same security domain, as in the case of Department of Defense (DoD), do not represent much of a problem since all the systems will support compatible labeling options. Cross-domain communications complicate the problem since labeling schemes could be incompatible. Translation of clearances and sensitivity categories may be possible by obtaining pairwise inter-domain agreements. This may require the intervention of a third party acting as a registration authority for labeling sets. Such an organization will provide guidelines for the definition and identification of security label sets so that an acceptable translation can be agreed upon. The National Institute of Standards and Technology (NIST) is considering providing such a service.

Already work has been done by organizations such as the Trusted Systems Interoperability Group (TSIG) in adapting the IPSO label to reflect the needs of the commercial sector. This proposed solution is referred to as the Commercial Internet Protocol Security Option (CIPSO). The CIPSO will be very influential in the process of standardizing security labels for OSI.

Security Labels Workshop

The National Institute of Standards and Technology hosted an invitational workshop called Security Labels for Open Systems. The scope this workshop went beyond security labels for Open Systems Interconnection (OSI) by looking at security labels in the more general context of open systems. NIST's main goal was to gather enough information from users and experts in network security as to draft sections on security labeling for Chapter 6 of GOSIP [8].

Among the attendees to this workshop were representatives from several DoD agencies, DoE, NIST, and companies such as Oracle Corporation, MITRE Corporation, Digital Equipment Corporation, Sears Technology Services, Xerox Special Information Systems, IBM, etc. A number of position papers were presented covering topics such as security policy, a DoE proposal for security labeling, OSI-based labeling strategies, CIPSO, security labels in database management systems, end-to-end encryption (E3), the Defense Message System, information identification and protection, labeling in open heterogeneous distributed systems, information labels, etc. In addition, NIST personnel discussed Security and the Portable Operating System Interface (POSIX), and Labels for Confidentiality, Integrity, and Availability.

One of the main issues discussed during this two-day workshop was whether or not confidentiality, integrity, and availability services should be handled by security labels. There was agreement in using labels within OSI to indicate integrity and confidentiality but not in regard to availability. Even though the value of the availability service was acknowledged it was argued that it does not belong in a network security label. The rationale for this is that no rule based access control (RBAC) decisions can be made based on an availability parameter. The

alternative is to rely on quality of service (QOS) parameters to handle this service. Billing codes, authorization and authentication mechanisms, and identity-based access control all of which had been discussed in other forums were also said to be out of the scope of security labels for the same lack of RBAC support.
The problem introduced by multiple security domains with incompatible sensitivity level and clearance definitions was also an important topic of discussion. Security labels directly reflect the definition of sensitivity levels. It seems that the use of a registration authority to address this problem is of general acceptance.

Towards the end of the workshop the group agreed to make several statements that would constitute its output. Those statements are presented below.

- The overall scheme for security labels should identify country versions for security labels.

Given that a unified labeling scheme for secure OSI would be presented to the international community as an U.S. contribution, provisions have to be made for distinguishing between label versions for different countries. This would be done by means of a Country-Version/Registration Authority field at the beginning of the label. Such a field would contain hierarchical information expanded to identify registration authorities.

- Options 130 and 133 (Basic Security and Extended Security Options) should be enhanced with the TSIG's Commercial IPSO options.

The IPSO based CLNP label already fulfills a number of basic requirements for security labeling. By merging this well established labeling scheme with the industry-developed CIPSO we can obtain a consensus standard for security labeling that will address the needs of the different user communities.

- SP4 and CLNP should use the same kind of security label.

By using the same kind of label at both layers 3 and 4 compatibility concerns could be eased.

- NIST should be the Registration Authority for security labels.

The use of registration authorities is necessary to allow the use of security labels tailored to a specific security domain and still be able to perform secure inter-domain communications. Given the neutrality of NIST and its responsibility for unclassified computer and network security the workshop attendees agreed that it should act as the U.S. registration authority.

- This group [the workshop attendees] should review sections on security labels added to GOSIP by NIST.

At NIST's request the workshop attendees agreed to provide expert review and comment on text to be drafted by NIST for inclusion in GOSIP.

The Next Steps

There has already been progress in the standardization of security labels for OSI and from the U.S. Government perspective the next step is to initiate the process of updating chapter 6 of GOSIP accordingly. New text is being drafted and will be available for expert review and comment shortly. As we have already mentioned the attendees to NIST's workshop on security labels have agreed to provide feedback on this text. The outcome of this work will also be presented to the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO).

After accomplishing this focus will be shifted to other areas such as key management.

References

[1] Brown D., Security Labels in TCP/IP Networks, Sandia National Laboratories, 1989.

[2] DDN Protocol Handbook, Volume One, DDN Network Information Center, December 1985.

[3] Dinkel C., Secure Data Network System (SDNS) Network, Transport, and Message Security Protocols, NISTIR 90-4250, February 1990.

[4] Housley, R., Security Labels in Open Systems: A Position Paper, in Security Labels for Open Systems: An Invitational Workshop, NISTIR 90-4362, pp. 83-84, June 1990.

[5] Linn, J., Commercial IP Security Option, in Security Labels for Open Systems: An Invitational Workshop, NISTIR 90-4362, pp. 117-121, June 1990.

[6] Nazario, N., Security Labels for Open Systems: An Invitational Workshop, NISTIR 90-4362, June 1990.

[7] St. Johns, Capt. M., RFC 1038: Draft Revised IP Security Option, DDN Network Information Center, January, 1988.

[8] U.S. Government Open Systems Interconnection Profile (GOSIP), Version 1.0, FIPS Publication 146, National Institute of Standards and Technology, June 1988.

KEY MANAGEMENT SYSTEMS COMBINING X9.17 AND PUBLIC KEY TECHNIQUES

Jon Graff, Ph.D.
Cylink
110 South Wolfe Road
Sunnyvale, CA 94086

(408) 735-5878

Proposed Key Management Protocols using Public Key and Symmetrical Key Techniques

Abstract

This paper describes a key management protocol that combines public key techniques with the symmetrical key techniques. The key management protocol standard for wholesale financial institutions, X9.17, serves as a basis for the proposed protocol. X9.17 uses manually delivered symmetric key encrypting keys to initially exchange keys. Subsequently, encryption keys, while encrypted under key encrypting keys, can be electronically transferred. The Cylink CIDEC-LS link encryptor's key management system serves as a basis for a an practical, initial model of incorporating public key techniques as a supplement to X9.17. The protocol permits the establishment of initial key encrypting keys using the Diffie-Hellman public key algorithm. The paper then discusses the further enhancements to achieve a key management system suitable for a dynamic network such as a Local Area Network (LAN). A recently proposed companion standard to X9.17 and a suggested method for Key Management to IEEE 802.10, SILS, have been developed from the concepts present in this paper. Additionally, the paper discusses the various properties of the available public key algorithms.

Introduction

Currently X9E9 and the LAN Security Working Group for IEEE 802.10, Standard for Interoperable LAN Security (SILS), are studying key management methods using public key techniques to establish mutually shared secret keys. This paper outlines one of the suggested approaches.

A Description of the Existing Key Management Protocols based on X9.17

X9.17, Wholesale Financial Institute Key Management

X9.17 is the Standard for Wholesale Financial Institute Key Management. It is published as the Financial Institution Key Management (Wholesale) X9.17-1985 by the American Bankers Association and is referred to as either ANSI X9.17 or X9.17. X9.17 is a key management system that uses a symmetrical keying algorithm (DES [1]) in a two level encrypting key system. The system is comprised of manually delivered key encrypting keys (KKs) that then permit the encryption of other keys (both KKs and traffic keys (KDs)) for their subsequent electronic distribution. The KDs are used singly to encrypt transmitted data, while the KKs are used in pairs (a pair of KKs is symbolized as *KKs).

The *KKs are 128 bits long and are composed of two 64 bit DES keys. The *KKs encrypt keying material by encrypting the keying material with the *KK's first DES key, then decrypting the result with the *KK's second DES key and finally re-encrypting the result with the *KK's first DES key. To retrieve the encrypted key material, the process is reversed; decrypting with the *KK's first key, encrypting with the *KK's second key and finally decrypting the result with the *KK's first key.

X9.17 requires that at least one initial *KK be manually distributed to each user (i.e., end encryption device). Subsequently, other keying material (both *KKs and KDs) can be exchanged over the public network encrypted under *KKs. The requirement for the initial manual transmission of secret information makes this system is susceptible to a "key purchase" attack or "spoofing".

Within X9.17, there are three methods for electronically exchanging encryption keys:

1. Direct, user to user: if the two users share a common *KK and one of them is capable of generating keys, they may establish a commonly shared KD between themselves as needed. The common *KK is used by one party to encrypt the traffic key which is then sent to the second party.

2. Indirect, user through the Key Distribution Center (CKD) to user: If the two users do not share a common *KK and neither has the facility to generate keys, but they individually share a *KK with the CKD, they may establish shared keys through the CKD. One of the parties asks the Key Distribution Center for a KD. The CKD generates 2 copies of the new KD, encrypting one under the first party's *KK and the other under the second party's *KK. Both these encrypted KDs are then sent to the first party, who subsequently sends the second encrypted KD to the second party. When both parties decrypt the new KD, they will share it and both will use it for traffic

encryption.

3. Indirect, user through the Key Translation Center (CKT) to user: if the two users do not share a common *KK but one of them has the ability of generating keys and each party possesses a *KK with the CKT, they may establish shared keys through the Key Translation Center. One user originates and sends *KKs or KDs to the CKT. The CKT then translates the keys (i.e. encrypts the keys) to a *KK that only the second user can read. Subsequently, the first user sends the encrypted keys to the second user to establish the keying relationship.

X9.17 meets the need for "peer entity authentication" (i.e. verification of with whom you are communicating) by requiring manually distributed initial *KKs as well as "notarization" of keys which occurs when the keys are transferred through a CKT or CKD. Possession of shared *KKs as well as process of CKT or CKD "notarization" of electronically delivered keys guarantees the mutual authenticity of the connected users. X9.17, however, does not protect users from repudiation.

One problem with X9.17 is that there is no provision for two parties to communicate if they do not share either a *KK between themselves or *KKs with a common CKT or CKD. The proposed ANSI Standard X9.28 "Multiple Center Key Management standard addresses this problem, and has been recently voted out for balloting by the X9E9 working group, the X9.17 parent committee.

The Cylink CIDEC-LS Key Management System

The Cylink CIDEC-LS link encryptor is an example of a practical key management system combining X9.17 with public key techniques. The system has been successful in use in major financial institutions for several years. The CIDEC-LS Key Management Protocol eliminates the need to manually distribute secret *KKs by using a variation of the Diffie-Hellman Key Exchange System ([2] and [3]), called "SEEK™" (Secure Electronic Exchange of Keys) to establish mutually shared secret *KKs. (The Diffie-Hellman algorithm will be explained in a later section of this paper. The section will describe the various public key systems.) Once the shared *KKs are established subsequent key exchanges are done using the faster X9.17 key exchange protocols.

The CIDEC-LS Key Exchange Commands

There are three types of command sets within the CIDEC-LS key management protocol:

1. The SEEK™ commands. These commands are used to establish the mutually shared, secret variable.

    a. request to establish a secret key using SEEK™. This command contains the initiating party's public key.

    b. response to the request to establish a secret key using SEEK™. This command contains the responding party's public key.

    Each party then calculates a shared secret number Z. In this implementation, the CIDEC-LS Key Management Protocol splits Z into several DES keys. Some of the keys are used as *KK pairs and the remaining pair of DES keys is saved for future authentication. This arrangement is arbitrary and Z may be split into KDs or any combination of *KKs, KDs or authentication variables as desired.

2. The symmetrical key negotiation commands. These commands are used to negotiate how to allocate the symmetrical (DES) keys derived from the shared, secret variable Z. In this application, using X9.17, this command set only specifies *KKs. If X9.17 is not used, this command can be used to negotiate KDs.

    a. request of a specific symmetrical key (KD) or key pair (*KK).

    b. response to the request for a specific KD or *KK. This response may be either positive or negative and permits the two units to negotiate which key to use and to align their key lists.

    The authentication keys are used to form a Message Authentication Code (MAC) in the next exchange of public key variables.

3. The X9.17 symmetrical key exchange protocol commands. X9.17 defines these messages. These messages are used to exchange KDs.

    a. Request Service Initiation (RSI): request to establish a KD.

b. Key Service Message (KSM): response with an encrypted KD.

c. Response Service Message (RSM): acknowledgement of correct receipt of the Data Key.

d. Error Service Message (ESM): response to a RSI.

e. Error Service Message (ESM): response to a KSM.

f. Error Service Message (ESM): response to a RSM.

Other CIDEC-LS Key Management Protocols Facilities

The CIDEC-LS Key Management Protocol, written for link encryptors, was designed for a static environment with dedicated communication pairs, although it can be used in a star configuration. Because the communication partners are fixed and known there is no facility for non-repudiation. However the protocol does call for out-of-band authentication to eliminate a "person in the middle" attack or "spoofing".

The out-of-band authentication takes place after the initial key exchange. An 8 bit hash is made of Y_A and Y_B. Each encryptor then displays the hash. The installers telephone each other and mutually verify their authenticity by voice recognition. Then one installer reads the beginning part of the display to the other and then the second installer reads back the last part of the hash. Because it is computationally infeasible for anyone but the two connected encryption units to show the correct displays, the procedure proves there is no "person in the middle'. Once installed only these two devices communicate; any properly encrypted messages received must have originated from the other partner.

The CIDEC-LS uses a self-synchronizing DES encryption mode and consequently this protocol has no facility for generating or sending cryptographic synchronization vectors.

Concepts for the Key Management Proposal

Key Management Requirements for a Dynamic Network

A dynamic network such as a LAN requires additional security features to those offered by the CIDEC-LS Key Management Protocols. In many LANs, users (i.e. end user devices) are frequently added and deleted, and the LAN itself may be frequently reconfiqured. Therefore, "message origin authentication" (i.e. the verified identity of who originated the message) becomes a serious concern. Public key techniques offer message origin authentication with digital certificates as well as protection from repudiation with digital signatures.

Electronic Digital Signatures

Electronic digital signatures protect the recipient from repudiation by the sender.

A digital signature consists of a piece of data encrypted in such a manner that only the sender could have encrypted it. The signature contains at least:

1. a hash that is dependent on the entire message. This hash is a publicly known function and its reproducibility by the receiver indicates that the message has not been modified in transit. This idea has been proposed as part of the authentication directory system in the Annex D of the CCITT Recommendation X5.09 (ISO 9594-8 The Directory - Authentication Framework).

2. a unique message identifier such as a time stamp or message sequence number to protect against replay

The signature is encrypted using the sender's secret key so that anyone can decrypt the signature using the sender's public key. Two methods for digital signatures are RSA [4] and ElGamal [5].

Digital Certificate

Although electronic digital signatures protect against repudiation and message modification, they do not guarantee the sender's authenticity. Proof of authenticity is supplied by a special case of signature called "certificates". Certificates originate from a trusted Certification Center. The Certificate Center system requires the user to communicate with the Certification Center only once during the life of the certificate. Once certified, a user may freely establish secret communication, without the assistance of the Certification Center, with any other certified user.

When first logging onto a network and then periodically, as required thereafter, each new member to the network applies for a Certificate from the Certification Center. This initial communication may be out-of-band or may be a secret conversation with the Center, possibly using a public key techniques. During this initial communication the new subscriber and the Certification Center mutually prove their identities to each other. This communication need not contain secret information; it need only contain the information required to assure the party's identity (for example a finger or voice print). However, this communication must be secure against modification in transit. (See [6] for a discussion of a possible scheme for authentication and identification.)

The Center then formulates a certificate that contains the new member's public key and other pertinent information about the member such as its identification number, privileges, address, and expiration date. The certificate is then encrypted using the Certificate Center's secret number. Henceforth, anyone on the network can decrypt the certificate using the Certification Center's public number. The user can attach to a message a signature, providing repudiation protection, and a certificate, providing data origin authentication. The message recipient now has the protection afforded by the signature and certificate, plus the added benefit of obtaining the sender's public key within the certificate, thus saving the time require to look it up.

A further extension of the certification concept establishes a hierarchy of Certificate Centers, with each higher center certifying its "children". This would greatly reduce the amount of work required by any one Certificate Center. This idea has been proposed as an authentication directory system in CCITT Recommendation X.500 (ISO 9594/1-8 The Directory - Overview of Concepts, Models and Services).

In contrast to a compromise of X9.17's CKD or CKT, a compromise of a Certification Center compromises only the validity of its certificates because the encryption keys, both *KKs and KDs, are generated independently by the end users. The compromise of X9.17's CKT or CKD compromises not only the identity of the system's users, but also all the traffic within the system.

<u>Certification Revocation</u>

To revocate a certificate, a Certification Center would broadcast a dated, signed message containing a list of compromised or not-to-be-trusted users whose certificates are suspect. Linn and Kent [7] suggests that the broadcast message contain the time of the next expected broadcast message to insure that none of these compromise lists are missed. The individual users would check the validity of their communication partners to confirm that they are not on this list. Additionally, each certificate has a finite life, requiring each user to periodically verify its identification with the Center.

Outline of the Key Management Proposal

The key management proposals require only one case for establishing a mutually shared key between two partners instead of the three described for X9.17. When a pair of users wish to communicate, they simply exchange authorization and authentication information and then establish a mutually shared *KKS and KDs using public key techniques. This exchange does not require any previous secret knowledge to be shared between users nor does it require the continuous assistance or availability of centralized key management.

As with the centralized key management, each individual user must have an initial contact with the central authority, or its delegate, to obtain a certificate. However, in contrast to the initial manual exchange in X9.17, this authorization need not be secret, only secure.

Based on the previous discussion the Key Management System must support four basic functions:

1. A procedure for logging onto the network by obtaining:

   A. a Certificate
   B. a Broadcast Key (required for LANs)
   C. Multicast Key(s) (required for some specialized uses in LANs)

2. A public key procedure for establishing initial secret keying associations between users. Neither proposal specifies which public key algorithm is to be used.

3. A symmetrical key exchange procedure for exchanging data (traffic) keys. The X9 proposal will use DES with *KKs to encrypt transmitted *KKs and KDs.

4. A procedure for facilitating encrypted traffic using the data keys. The X9 proposal will use DES.

These requirements can be met by:

1. Establishing a hierarchical Network Management system that provides Certificates and the necessary common multiuser keys. The initial contact between users and the Network Management might be through public key techniques although at this time an out-of-band communication seems most practical. Additionally, provisions are needed for both updating and revoking certificates.

2. Initial contact between entities within the network would be through a defined public key technique (s), thereby exchanging or mutually developing shared secret keys.

3. The conventional (symmetrical) encryption technique to encrypt traffic. This is currently within the X9.23 Wholesale Banking Message Encryption standard and supplement to X9.9 Wholesale Banking Message Authentication standard.

4. Definitions and procedures for digital signatures and certificates would have to be specified. This requires public key techniques and would be used in the key exchange process. These techniques would be available at the Application Layer for other uses such as digital signatures and certificates for fund transfers and contracts.

Description of the Public Key Algorithms

Table 1 presents a summary of the properties of the three major public key algorithms. The mathematics for each algorithm are shown in the Figures 1 through 5. The Diffie-Hellman algorithm is described in only one figure because it is mainly applicable for establishing a secret key between users and not for signatures. Both the RSA and ElGamal algorithms have two figures a piece because they are readily applicable for both sending messages and signatures, see Table 1.

As shown in Table 1, the three algorithms differ in the requirement of mutual participation in establishing a shared secret key. The Diffie-Hellman algorithm requires the mutual participation of the parties to establish a common key. (As an aside, it is possible to establish a common key among several user with the Diffie-Hellman algorithm. Each user simply submits its public number to the collective pool and then each user exponentiates the other's public number to calculate the commonly shared number "Z".) The ElGamal and RSA algorithm do not require the mutual participation in the establishment of a mutually shared secret key. These two algorithms permit one user to unilaterally send the second a secret number. This secret number is only received by the second party and the second user is not responsible for its selection. However, it is possible for the two parties to mutually calculate a shared secret number with the ElGamal and RSA algorithms. The mutual established key requires that the second user to send an additional message containing a second secret number to the first user. Then each of the users calculates the commonly shared secret number based on the two newly exchanged secret numbers.

The three algorithms also differ in their ability to produce different ciphertext with each exchange. With the proper implementation of the ElGamal algorithm each encrypted message or signature is random. RSA encryption and signature can also produce random ciphertext from the same plaintext but this requires that a unique character string must be appended to the plaintext message. The Diffie-Hellman algorithm can also supply different "public key" numbers for each key exchange. Each user selects a new pair of secret and public numbers solely in each exchange, but maintains its "permanent" secret and private key pair for signature purposes (using ElGamal signatures). The calculations would be exactly that shown in Figure 1, but with each variable having an "$_i$" to show that the variable only exists for this particular key exchange. For example, Alice selects for this exchange a one-time random secret number S_{A_i} and calculates a new public number P_{A_i}. Alice sends P_{A_i} to Bob in a message that Alice signs with an ElGamal signature based on Alice's permanent secret and public numbers S_A and P_A and her certificate, if required. Similarly, Bob would reply with the one-time public number P_{B_i} calculated from the one-time secret number S_{B_i}. Then Alice and Bob calculate the shared secret number Z_i.

The algorithms differ in the number of messages required to establish a shared secret key. The Diffie-Hellman algorithm only requires two messages to establish a mutually shared and computed secret key. Additionally, the Diffie-Hellman algorithm does not require the prior knowledge of the recipient's public number. For instance, if Alice wishes to establish a mutually computed shared secret number with Bob, she computes a one-time public number S_{A_i} then composes a message containing S_{A_i} signed with her permanent public-secret key pair (P_A, S_A) and her ElGamal certificate. If Bob wants to establish the keying with Alice, he responses with a message containing his one-time public number S_{A_i} signed with his public-secret key pair (P_B, P_B) and his ElGamal certificate. Then Alice and Bob can mutually compute their shared secret number Z_i.

The RSA and ElGamal algorithms require at least three messages to establish a shared secret key and four to establish a mutually computed shared secret key. First Alice must determine Bob's public number. This query could be sent to one of a number of places, for instance a central data base containing certificates or to Bob requesting Bob's certificate. Regardless of the source, the second message, i.e. the response, would be Bob's certificate. Then Alice would compose and send to Bob a third message containing Alice's one-time secret number encrypted with Bob's public number, Alice's signature and certificate. If it was desired to establish a mutually computed shared secret number, Bob would reply with a fourth message composed of his one-time secret number, Bob's signature and certificate. Then Alice and Bob would mutually calculate a shared secret number based on their two one-time secret numbers.

Discussion

Besides the Cylink CIDEC LS modification of the X9.17 key management system, there are at least four key management systems that have been reported in the literature [7, 8 and 9, 10, 11] having similar two tiered key systems. The key encrypting keys are initially constructed or exchanged using public key techniques. Data is then encrypted using symmetric traffic keys.

The key management systems differ on what algorithm they use to exchange keys. The SDNS [8 and 9], uses a secret algorithm called FIREFLY for its key exchange and authentication. DARPA Internet Mail [7] and the Digital Distributed System Security Architecture [10] use RSA as the public key technique. The CIDEC-LS system and MEMO (in the non-PKF approach) [11] use the Diffie-Hellman technique to construct and exchange its key encrypting keys. The proposed companion standard to X9.17 is, at least at this time, algorithm independent.

SDNS, the DARPA Internet Mail and the proposed companion standard to X9.17 use a broadcasted revocation list to notify users of invalid certificates. The DEC system revokes certificates by omission, i.e. invalid users are deleted from a list of users having permission to access a process and the entity offering a service must verify that a user is on its permission list before performing the requested service.

Conclusions

This paper presents a practical implementation of a key management system for link encryptors that successfully combines public key techniques with the wholesale financial standard, X9.17, which uses symmetrical key techniques. The key management requirements for a dynamic network are discussed. The paper then describes the key management system proposed in the proposed companion standard to X9.17 that uses combined symmetrical and public key techniques. The three available public key algorithms are compared. The Diffie-Hellman algorithm is the best suited for establishing a mutually calculated shared secret key. The ElGamal and RSA algorithms are best suited for the calculating digital signatures and certificates. The methods for producing variable ciphertext for each of the algorithms are discussed.

References

[1] Data Encryption Standard (FIPS PUB 46), National Bureau of Standards, January 1977.

[2] W. Diffie and M.E. Hellman, "New Directions in Cryptography," in IEEE Transactions on Information Theory, Vol. IT-22, pp. 644-654, November 1976.

[3] W. Diffie, "The First Ten Years of Public-Key Cryptography," in Proceedings of the IEEE, Vol. 76, pp. 560-577, May 1988.

[4] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," in Communications of the Association of Computing Machinery, Vol 21, pp. 120-126, February 1978.

[5] T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," in IEEE Transactions on Information Theory, Vol. IT-31, pp. 469-472, July 1985.

[6] M. Smid, J. Dray and R.B.J. Warnar, "A Token Based Access Control System for Computer Networks," in The Proceedings of the 12th National Computer Security Conference, pp. 232-253, 1989.

[7] J. Linn and S.T. Kent, "Privacy for DARPA-Internet Mail," in The Proceedings of the 12th National Computer Security Conference, pp. 215-229, 1989.

[8] R. Nelson, "SDNS Services and Architecture," In The Proceedings of the 10th National Computer Security Conference, pp. 153-157, 1987.

[9] P.A. Lambert, "Architectural Model of the SDNS Key Management Protocol," in The Proceadings of the 11th National Computer Security Conference, pp. 126-128, 1988.

[10] M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson, "The Digital Distributed System Security Architecture," in The Proceedings of the 12th National Computer Security Conference, pp. 305-319, 1989.

[11] B. Schanning, S.A. Powers, and J. Kowalchuk, "MEMO: Privacy and Authentication for the Automated Office", In the Fifth Conference on Local Computer Networks, pp. 21-30, 1980.

Table 1: A Brief Description of the Publicly Available Public Key Algorithms

Public Key Technique	Diffie-Hellman	RSA	ElGamal
Key Exchange Capability	yes	yes	yes
Signature Function	no	yes	yes
Generation of strong primes required for each public-private key pair	no	yes	no
Mutual party participation in secret key formation	yes	no	no
Random encryption in each cryptographic exchange	no	no	yes

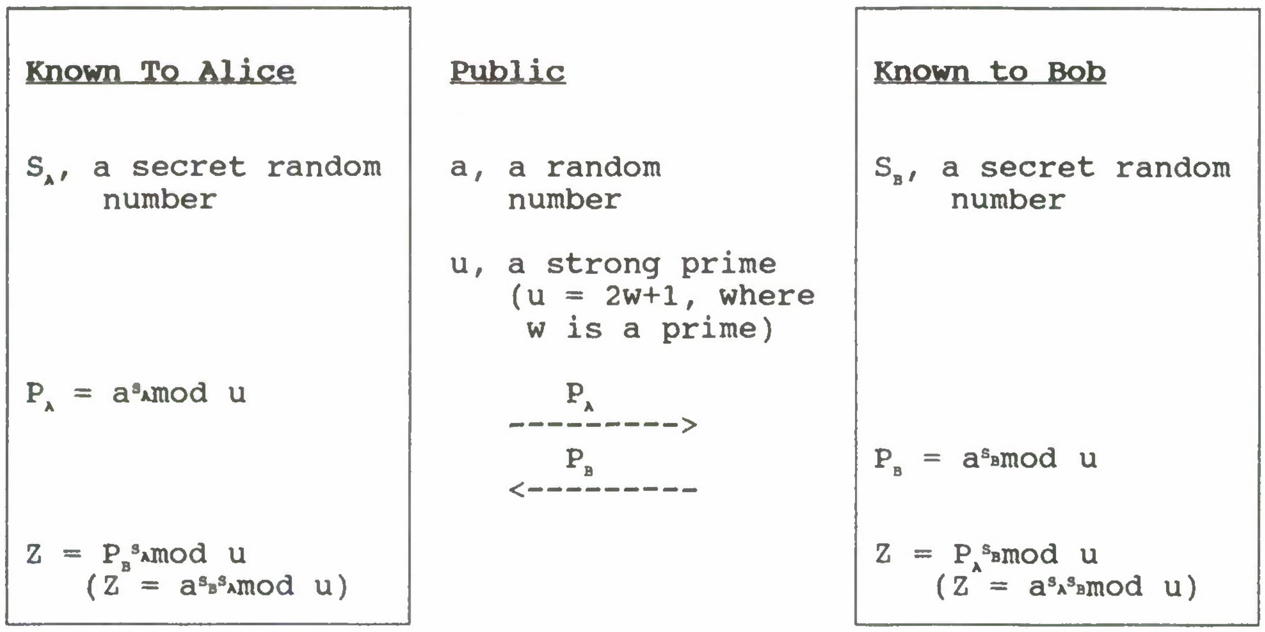

Known To Alice
S_A, a secret random number
$P_A = a^{S_A} \bmod u$
$Z = P_B^{S_A} \bmod u$
$(Z = a^{S_B S_A} \bmod u)$
Public
a, a random number
u, a strong prime (u = 2w+1, where w is a prime)
P_A
P_B
Known to Bob
S_B, a secret random number
$P_B = a^{S_B} \bmod u$
$Z = P_A^{S_B} \bmod u$
$(Z = a^{S_A S_B} \bmod u)$

Figure 1: **Diffie-Hellman Algorithm:** Alice and Bob mutually establish a secret shared key.

Known To Alice	**Public**	**Known to Bob**
	a, a random number	S_B, a secret random number
	u, a strong prime (u = 2w+1, where w is a prime)	
	$\xleftarrow{\quad P_B \quad}$	$P_B = a^{S_B} \bmod u$
r_1, a secret random number		
$v_1 = a^{r_1} \bmod u$	$\xrightarrow{\quad v_1 \quad}$	
$Z_1 = P_B^{r_1} \bmod u$ M_1, the message C_1, the ciphertext		$Z_1 = v_1^{S_B} \bmod u$
		$Z_1{*} = Z_1^{(w-2)} \bmod (u-1)$
$C_1 = M_1^{Z_1} \bmod u$	$\xrightarrow{\quad C_1 \quad}$	$M_1 = C_1^{Z_1{*}} \bmod u$

Figure 2: ElGamal Message Encryption Algorithm: Alice sends an encrypted message to Bob.

Known To Alice	Public	Known to Bob
S_A, a secret random number	a, a random number	
	u, a strong prime ($u = 2w+1$, where w is a prime)	
$P_A = a^{S_A} \bmod u$	$\xrightarrow{P_A}$	
r_i, a secret random number		
$v_i = a^{r_i} \bmod u$	$\xrightarrow{v_i}$	
M_i, the signed message	$\xrightarrow{M_i}$	
h_i, the hash of $(M_i \| v_i)$		
$h_i = H(M_i \| v_i)$		$h_i^* = H(M_i \| v_i)$
$Sign_i = r_i + h_i S_A \bmod u$	$\xrightarrow{Sign_i}$	
		Bob verifies that:
		$a^{sign_i} \bmod u = v_i P_A^{h_i^*} \bmod u$

Figure 3: **ElGamal Signature Algorithm:** Alice sends a signature ($Sign_i$) for message M_i to Bob.

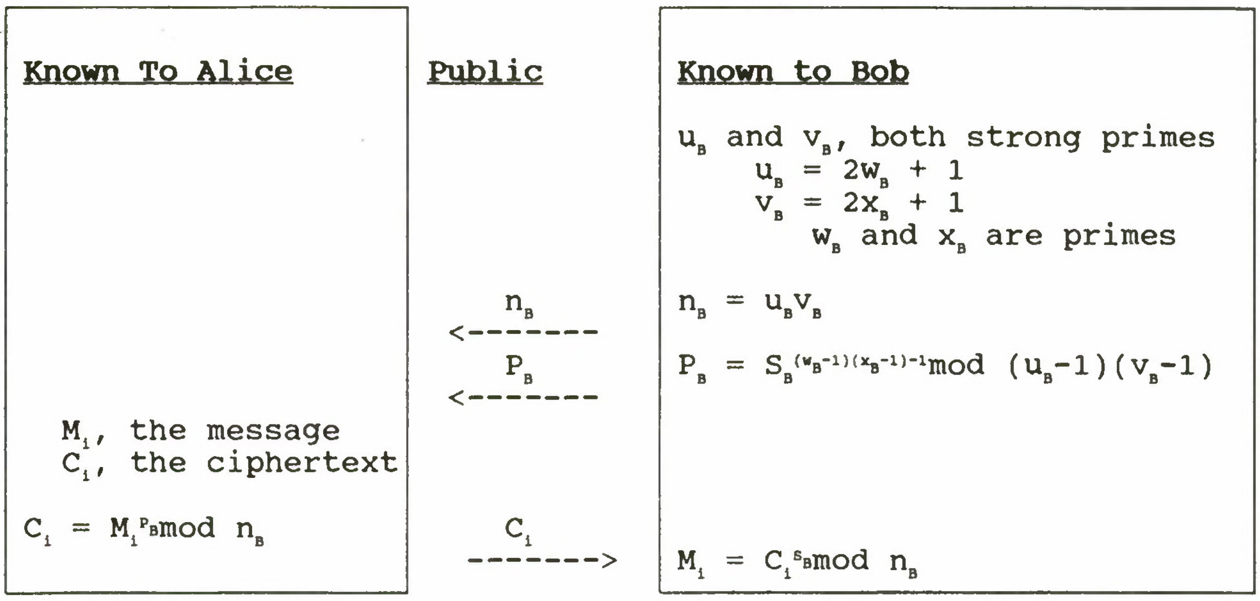

Known To Alice
Public
Known to Bob
u_B and v_B, both strong primes
$u_B = 2w_B + 1$
$v_B = 2x_B + 1$
w_B and x_B are primes
$n_B = u_B v_B$
$P_B = S_B^{(w_B-1)(x_B-1)-1} \bmod (u_B-1)(v_B-1)$
n_B
P_B
M_i, the message
C_i, the ciphertext
$C_i = M_i^{P_B} \bmod n_B$
C_i
$M_i = C_i^{S_B} \bmod n_B$

__Figure 4__: **RSA Encryption Algorithm:** Alice sends an encrypted message to Bob.

Known To Alice	Public	Known to Bob
u_A and v_A, both strong primes $u_A = 2w_A + 1$ $v_A = 2x_A + 1$ w_A and x_A are primes		
$n_A = u_A v_A$	n_A ------>	
$P_A = S_A^{(w_A - 1)(x_A - 1) - 1} \bmod (u_A - 1)(v_A - 1)$	P_A ------>	
M_1, the message	M_1 ------>	
h_1, the hash of M_1 $h_1 = H\ (M_1)$		$h_1{*} = H\ (M_1)$
$Sign_1$, the signature		
$Sign_1 = h_1^{S_A} \bmod n_A$	$Sign_1$ ------->	$h_1 = Sign_1^{P_A} \bmod n_A$ $Sign_1$ verified if: $h_1 = h_1{*}$

Figure 5: RSA Signature Algorithm: Alice sends a signature for message M_1 to Bob.

ELECTRONIC DOCUMENT AUTHORIZATION

Addison M. Fischer
Fischer International Systems Corporation
4073 Merchantile Avenue
Naples, Florida 33942

Abstract

This paper discusses an implementation of Electronic Document Authorization (EDA), a workable methodology for managing authority in a distributed environment. This methodology may be applied to the exercise and delegation of generalized authority. This paper especially focuses on specialized authority for money-responsibility.

EDA is a protocol using RSA public key digital signatures, which allows any electronic material, including that conforming to various EDI (Electronic Data Interchange) data formats, to be "provably authorized" based on the prima facie contents of the data itself.

The EDA protocol allows a document or file to be digitally authorized so that any recipient is able to prove to themself, or to others: the identity of the signer(s); whether the signer(s) actually had adequate authority within their organization to perform the authorization; and whether the signer(s) are in compliance with constraints their organizations may have imposed on them.

This methodology provides comprehensive authorization, verification, and authentication support to enhance EDI processing. It permits the full use of automated digital systems for creating, authorizing, distributing, receiving, validating, and otherwise processing electronic documents. Once the initial global "trust criteria" have been established, EDA validation may then be tested automatically by computer software.

EDA overcomes certain inherent weaknesses that occur in relying on a digital system to supplant conventional paper-based transactions — for example, it can reduce the exposure even when an encryption key is compromised.

In addition to being immediately applicable to value-related EDI, the EDA methodology is also designed for specialized authority needs unique to particular organizations. EDA allows organizations to define distributed, built-in, safeguards to forestall the possibility of corruption, fraud, misdirection, or other misuse or misrepresentation of the organization's resources. Such safeguards may be implemented in a multilevel fashion as deemed appropriate by the organization.

Background

EDI — Electronic Data Interchange — is a rapidly emerging technology which allows automated commercial transactions to be conducted within organizations and among enterprises. Using EDI, a business document can be created as an electronic file in the sender's system, sent via any of several possible transmission modes, and processed directly by recipients' computers. Among the potential benefits of this technology are cost savings, information accuracy, and improved timeliness (to name only a few).

Actually, the technology needed to implement EDI is here today. However, among the major obstacles to its widespread application has been a lack of security features — including authentication, non-repudiation, authorization, and provability. Because security is lacking, most uses of EDI to date seem to be between well-known trading partners across secure channels.

To fully realize the potential of EDI, some tough issues need to be resolved: How can we guarantee the accuracy and enforceability of EDI transactions such that they provide a natural substitute for conventional paper-based transactions between all business concerns? How can EDI transactions be validated when they are transferred across unsecure, or questionably secure channels? How can recipients of an EDI document prove — either to themselves, to others within their organization, or to someone outside their organization — whether a received document is authentic; who authorized the document; and whether the authorization was performed according to the guidelines and constraints dictated by the originating organization or by some third party such as a government agency?

As EDI becomes more widely used, so does the opportunity for misuse and corruption. As the population of EDI users increases, more and more traffic will pass between organizations which have dissimilar backgrounds, security needs, and security controls.

This paper focuses primarily on how EDA (Electronic Document Authorization) is used in conjunction with commercial EDI. However, the techniques developed for EDA are general in scope, and are not limited to EDI. EDA may be readily applied to any environment or situation in which it is useful to administer authority over digital material, so that authorization can be immediately verified on a prima-facie basis.

Motivation for EDA

EDA offers workable solutions to the problems involved with authentication and management of authority in a distributed environment. EDA brings the following features to the EDI environment:

- EDA enables adaptation of current business practices to entirely digital techniques.
- EDA fully distributes authentication and authorization across systems and networks of varying degrees of inherent security. For purposes of this article, **authentication** means verification of the identity of a communicating party, or validation of a communication. **Authorization** is permission, granted by a properly appointed person or persons, to perform some action.
- EDA provides full and provable responsibility and accountability for all authorizations.
- EDA minimizes the shared trust/knowledge necessary between recipient and sender. It requires no contractual or ongoing business relationship between sendor and recipient.
- EDA's basis in public key technology allows document authorization to be **proved** based solely on the digital contents of the document and its EDA seal.
- EDA allows document correctness, authenticity, and authorization to be proved without presuming continuous, unbroken access control.
- EDA allows document signatures and authorization to be proved at any future time, by any party, if a dispute or question should ever arise.
- EDA provides inherently strong safeguards to reduce the possibility of corruption, or other misuse, to whatever levels an organization deems appropriate. It allows a variety of safeguards to enable appropriate security treatments for differing risks.
- EDA allows each organization to regulate, tailor, and administer their own internal security controls in whatever manner they deem appropriate. EDA allows organizations to delegate control in a fully distributed manner, permitting appropriate safeguards to be applied at each step.
- EDA allows received documents to be validated automatically, by computer.
- EDA is not tied to any particular framework. It applies to any digital file or document, independent of format or contents.
- EDA provides upward compatibility with X.509 standards. (X.509 relates to implementation of security measures in electronic directory services.)

Conventional business practices generally include certain built-in safeguards which are implemented and evaluated by persons having specific responsibility and authority. For example, a purchase order is usually produced on an "official" form printed with the issuer's logo. Purchase order forms themselves may bear serial numbers and may be stored under lock and key. As part of the requisition process, the purchase order is signed by an individual — possibly by an individual recognized by the recipient. Often it is signed by several persons, in accordance with the rules of the issuing organization. The document is likely to be stored, at least until the entire business transaction, including payment, is complete. Finally, it is likely to be archived for some period thereafter, in case some dispute or question regarding the transaction should ever arise.

In the brave new world of EDI, many of these safeguards are lost. To start with, in principle, a digital file containing any data can be created by anyone at any time. With EDI, the pre-printed company form, and the handwritten signature are gone.

In most commercial EDI applications, security is rudimentary: the recipient must trust that the document was honestly sent, and received over the selected media (network, floppy, etc.). Furthermore, once the document arrives, it is up to the recipient to ensure that it is safely stored under adequate access control at all times (to guard against tampering by internal personnel).

EDA is a single, comprehensive methodology that answers major concerns arising from EDI's lack of inherent security. EDA provides authentication and authorization that is fully compatible with EDI's digital format. As a side benefit, the security of transmission and storage methods are rendered irrelevant.

Although the implementation of EDA digital authorization is strongly analogous to paper authorization, it is not identical. In many respects it provides stronger proof of authorization than paper would.

One of EDA's guiding principles is a recognition of the truth that individuals wield power and authority on behalf of organizations. EDA also realizes that individuals are unpredictably fallible, and in some cases, corruptible.

The Technology Behind EDA

EDA is a protocol which uses RSA Public Key Digital Signature technology. Because RSA public key technique is fundamental to understanding EDA, this section briefly describes its history and essential properties. Readers who desire more detailed information may see the ***References*** cited at the end of the paper.

The concept of public key technology was first proposed by Whitfield Diffie and Martin Hellman at Stanford University in 1976. Diffie and Hellman did not produce a working public key system, but less than a year later, the RSA public key system was invented at MIT by Professors Rivest, Shamir and Adleman. Although a number of other various public key techniques have been proposed, most of them have quickly fallen by the wayside. Only RSA has withstood over a decade of intense scrutiny. RSA has already been accepted, or is in the process of being accepted, by a number of standards committees worldwide. Where it has not already been made the official standard, it has become the de facto standard.

RSA has important implications for security in many different areas, including data privacy (encryption), data integrity, and authentication. Although many aspects of the RSA public key system are of interest from a security perspective, we will confine ourselves to the facets of the system relating to digital signatures.

RSA public key technology is based on the creation of two large numeric values known as the "public key" and the "private key," which are related under special mathematical operations in remarkable ways: Performing the "signature" operation with the private key on any arbitrary digital value "A" produces a result "S" (the signature). Once this "S" value is created, anyone can perform the "verification" operation on "S" using the public key and get the original signed value "A" as the result.

What makes this special is that the signature value "S" can only be computed using the private key. The signature "S" is a number hundreds of digits long. Given any particular message, there is one and only one signature value for any public/private key pair.

The strength of RSA is that the signature value can *only* be computed using the private key. Knowing the public key provides no help whatever in determining the value of "S". However, once the value is known, the public key will easily ***verify*** it. Another way of saying this is that the public key operation is not "invertible" – i.e., given an arbitrary document, there is no way to "run the ***public*** key operation backwards" to compute the signature value; it can only be computed with the private key.

Other important properties of RSA Digital Signatures are these:

- The slightest alteration of any kind in either the signature or the signed data causes the verification process to fail.
- Given any file (or any signature), it is equally impossible to find any different file that leads to the same signature.

(Those who wish to pursue the mathematics further may consult the ***References*** cited at the end of this paper. Although the mathematics is not simple, it involves no calculus, and only elementary number theory.)

Application of RSA to EDA

The upshot of RSA Digital Signature operations is this: I can digitally sign any file. If everyone knows what my public key is, then anyone can verify the signature and conclude that the signature was produced only by the holder of the private key – namely, me. Furthermore, I can be assured that my digital signature can never be applied to any data without the use of my private key.

The preceding statement defines the powerful capabilities of RSA technology. But these capabilities alone cannot meet the security-related needs of EDI – needs which the EDA protocols solve.

Some of the more obvious problems include:

- Even if a recipient is convinced that a signer is accurately identified, how can the recipient be assured that the signer is acting within the scope of his authority?
- How can an enterprise ensure that authority is properly controlled, administered, and executed throughout their organization?
- How can a recipient trust that a signer's public key actually belongs to that signer?
- Because digital signatures depend on the confidentiality of private keys, how can an organization ameliorate the danger of a private key being revealed (accidentally or otherwise)?

EDA provides security in the face of the new challenges posed by EDI. As inter-enterprise document handling becomes more automated, and human scrutiny is reduced (or eliminated), the opportunity for new forms of mischief increases. Digital verification techniques need to be suitable for computer checking, reliable, as failsafe as appropriate, and effective.

The Structure of EDA

Every signature used in EDA is accompanied by an "authorizing certificate" (which we may call either an "EDA authorization," or an "EDA certificate").

Each EDA certificate identifies:

- The public key associated with the signer's private key.
- The name of the associated user.
- The organization of the associated user.
- Other optional identifying information.

Each EDA certificate also specifies the authority which is granted to the user, and the limitations and restrictions on this authority which have been placed by the organization on the user. Each EDA certificate defines the following authorities granted (if any):

- The ability to authorize expenditure ("money").[1]
- The ability to further identify other users.

Each EDA certificate defines the following restrictions/limitations:

- The expiration date of the certificate.
- The maximum amount of money which may be authorized by this user for any given transaction.
- Whether, and to what extent, each of the authorities may be further delegated.
- A set of co-signers, whose digital signatures are required on any object signed under authority of this certificate before any digital signature performed under this certificate may be considered fully authorized.

This last restriction allows organizations to define and enforce checks-and-balances as part of their underlying authority structure. For authority in matters of unusual gravity or far-reaching effects, it can ensure that no single user is able to take unilateral action. Co-signature requirements can be null (with no requirement), simple (with only a single co-signer), or quite complex (with different groups of possible co-signers, from which various subsets may be used to satisfy the requirement).

In performing an EDA signature, a user specifies the certificate (if he possesses several of differing characteristics) he intends to use. This certificate is then incorporated into the signature data, so that its authorizations (and limitations) become inherently bound into the EDA signature value. No one will be able to verify the signature without having both an unaltered copy of the signed data, and an unaltered copy of the certificate.

[1] Although financial or fiduciary authority is used throughout this paper as an example of the kind of authority that may be defined by a certificate, other types of authority could just as well be specified (e.g., authority to commit troops or to release classified information in a military scenario).

(Henceforth, we will frequently speak of a signature being performed "by a certificate." It should be understood that this actually refers to a digital signature which is performed by the private key associated with the public key which is named in that certificate.)

In general, a certificate itself is merely another digital object which has no intrinsic value. A certificate obtains validity in two ways: by being signed by other certificates which delegate authorization to it, or by being "universally" recognized and accepted.

The first type of certificate, known as a *regular* certificate, must be signed by other users with sufficient aggregate authority (as witnessed by their own respective certificates and restrictions) to properly grant the authorities. Regular certificates derive their authority through delegation from a higher level.

The second type, known as *meta* certificates, are not signed by other certificates. These certificates derive their "authority" from the fact that they are well-known and usually well-publicized. They must be directly recognized by participants in the EDA population.

The primary duty of a meta-certifier is to accurately certify the top-level keys associated with participating EDA organizations. In a sense, the meta-certifiers act as the ultimate "glue" which binds together EDA participants.

Meta-certificates can be subject to the same type of restrictions and safeguards as any other authorizing certificate. In the interests of overall reliability and trust, our recommendation is that *meta-certifiers be subject to co-signature requirements at least as stringent as any to be found in any organization.* This way, not even a single high-level meta-certifier can corrupt the system — either deliberately or inadvertently.

Explicit Delegation

The concept of explicit delegation is a feature of EDA which allows controlled distribution of authority throughout a hierarchy. For each authority class (of which there are presently two explicitly defined — the ability to authorize money expenditure, and the ability to further identify other users) there are four possible delegation levels that may be assigned (1 through 4). These levels are named (NONE, DEPUTY, OFFICER, MASTER) and have the following attributes:

NONE(1) The authority may be exercised by the user to the extent it was granted. It may not be delegated to other persons' certificates.

DEPUTY(2) The authority may be exercised, and the user may also delegate its exercise; however, the user is not permited to sub-delegate further sub-delegation authority.

OFFICER(3) The authority may be exercised, its exercise may be delegated, and Deputy sub-delegation authority may be granted. However, Officer authority may not be created.

MASTER(4) The authority may be exercised and further delegated as the user sees fit. This allows possible sub-delegation to any number of levels.

This delegation scheme allows exercise of authority to be granted, while managing the risk of losing control of the authority. The Deputy level allows delegation of exercise, without raising the question of whether a Deputy has the proper perspective to further judge the wisdom of others. The distinction between the Officer and the Master is possibly slight, but the distinction has been made available. However, further gradients between the Officer and Master seem to be pointless.

Money Authority Specification

Certificates may have an indefinite number of distinct "money authorizations." Each money authorization has three segments: *currency*, *limit*, and *delegation.*

This defines a particular currency[2] and the maximum value (amount) which may be specified in the digital signature by this user. The degree to which money authority may be delegated, if any, is specified by the "delegation." Certificates may, of course, be created without money authority.

[2] Currency of all nations is supported as regards ISO 4217.

Certificates may be created with a single money authority, e.g.:

```
  (USD, 500, No Delegation)
```

would allow a user to directly authorize 500 U.S. dollars on his EDA signature. No delegation is allowed.

Whereas, a certificate with:

```
  (CAD, 500, No delegation)
  (CAD, 200, Deputy)
```

would allow $500 Canadian direct authorization, and the ability to authorize other certificate holders to exercise up to $200 Canadian.

A user in a multinational corporation might have a multiplicity of authorities for various currencies:

```
  (USD, 10000, No delegation)  /* U.S. Dollars */
  (CAD, 12000, No delegation)  /* Canadian Dollars */
  (GBP,  8000, No delegation)  /* British Pound Sterling */
  (FRF, 40000, No delegation)  /* French Franc */
  (DEM, 30000, No delegation)  /* Deutsche Mark */
```

Identification Authority

EDA allows control of the authority to identify users on behalf of an organization.

The power to "Identify other users" is the authority to create certificates for them. The identifier also has the primary responsibility for cancelling its certificates should the need arise (this is further discussed later).

An installation may either grant or deny this authority. If an installation allows Identification authority, then it may (or may not) also choose to allow delegation in accordance with the general delegation rules. This leads to 5 different levels of identification authority:

0 No Authority No sub-Identification is permitted.

1 Identification The user may create certificates only with NO Identification Authority. The user is trusted to identify individuals, but not to judge whether they can be trusted to perform identification.

2 Identify/Deputy The user may create certificates with simple Identification(1) authority, but not with delegation authority.

3 Identify/Officer The user may create certificates with up to Deputy authority.

4 Identify/Master The user may create certificates with any delegation authority.

Co-signatures

When a certificate is created, it specifies the authorizations which are granted to the associated user. However, just as important, the certificate is also constructed with co-signature requirements. These requirements name other persons who must exercise their own digital signatures to "ratify," or "approve," any material authorized through use of the certificate. This ensures (subsequent) verifications will be aware of what other signatures are necessary before signed material is to be considered authorized.

A co-signature requirement is a list of zero or more items (zero, of course, indicating the absence of a co-signature requirement) together with a number specifying the number of items which must be satisfied. Each item may be one of three things: a reference to a public key, a reference to a certificate, or an embedded co-signature list (another list of items with its own satisfaction count).

The ability to inherently specify and enforce co-signatures is a strong and flexible protection with many benefits:

- It is a digital analog to the time-honored tradition of multiple "paper" signatures.
- Because digital signatures are always accompanied with their underlying certificate, any recipient will be able to instantly confirm (or not) that the signer's corporate policy has been fulfilled. This confirmation provides strong assurance that the authorization is trustworthy and can be acted upon.

- By requiring multiple co-signatures, individuals cannot make unilateral decisions. This substantially reduces the possibility of policy violations, misuse of authority, economic mischief, computer fraud, embezzlement, and other forms of corruption — generally before they can ever happen. Such acts will require collusion among the co-signers.
- It provides effective controls over the organization's resources.
- It provides enforced auditing of exercise of the user's authority. Several users, possibly on different platforms, in different geographic locations must concur on a particular authorization in order to make it effective. ***Checks and balances ensure that corporate policy is followed.***
- If a user compromises the password to his private key, then co-signature requirements substantially reduce risk of misuse, since other users are always required to concur. ***This is true even if the compromise is never detected.***
- It allows access-control security risks within an organization to be distributed across several hardware platforms, perhaps in different locales, governed by different personnel. Even if security is breached at one location, other systems are apt to remain uncorrupted. The impact of vulnerabilities on one platform are diluted.
- With fully distributed security, risks are substantially reduced.

A basic co-signature list might look like this:

```
  1 of the following are required:
    Joe's public key hash:          568AB678 AF317CEF 756301F6 5518891A
```

A simple co-signature list might be:

```
  2 of the following are required:
    Joe's public key hash:          568AB678 AF317CEF 756301F6 5518891A
    Bill's certificate hash:        0A37D687 46E7436A 8763E876 287D687E
    Sue's certificate hash:         7E2D36C8 A35E821B 537C2A38 6A3D21E7
```

A more complicated example with a nested list might be:

```
  3 of the following are required:
    Joe's public key hash:          568AB678 AB317CEF 756301F6 5518891A
    Bill's certificate hash:        0A37D687 46E7436A 8763E876 287D687E
    Controller's sublist:
     2 of the following are required:
        Sue's certificate hash:     7E2D36C8 A35E821B 537C2A38 6A3D21E7
        Bob's certificate hash:     64765457 56418765 87165815 47174657
        Sam's certificate hash:     D583A87F 7E82582C 7E287A78 2B872681
        Jill's certificate hash:    E87342D2 832D72C6 74A6276A 7825B216
        Dot's certificate hash:     346D7D16 78A16875 C2C2C687 A873B753
```

which would require that Joe, Bill, and at least two of the controller's staff must sign before the signature is valid.

<u>Signing with an EDA Certificate</u>

Invocation of authority is explicit: In signing an object which requires authority, a user must explicitly indicate the authority which is bestowed. In signing an Electronic Purchase Order for $325, for example, that amount (at least) must be stipulated at the time of signing. If the user is signing to delegate money powers to another certificate, then he must so state that he is invoking his authority.

A signature is not ***ratified*** until all co-signature requirements are satisfied. If the user signs an object (a file, an EDI document, or possibly another certificate), and the user has **no** co-signature requirements, then the signature is immediately ratified. However, if the user's certificate stipulates signatures by other parties are required, then the signature remains in an ***unratified*** state until sufficient signatures have been obtained to satisfy the requirement(s).

Contents of an EDA Signature Proof Packet

Whenever an object (file, certificate, etc.) is signed, the EDA signature information is typically carried as a separate object (file, or record).

In addition to the new information created by the private key operation, the EDA signature information contains the certificate associated with the signature, together with the signatures and certificates which "prove" the certificate (show that it is authorized). The proof-information for each of these, in turn, is also included. This hierarchy stops with the meta-certificates.

The EDA proof packet is condensed so that the implicit tree-structure described above does not contain duplication. In practice, the EDA proof packet contains only about 4 to 11 certificates and signatures, although the number could increase depending on the complexity and depth of the counter-signature rules which an organization wishes to use. It could be as few as two, in the simplest case.

Acceptance Criteria

Each person that verifies documents defines the meta-certificates which they choose to accept as valid. Meta-certificates, like all certificates, are computer records, containing a public key, rules, restrictions requirements, flags, and other data. In its raw form, this is not an easy object for a human to verify — especially since any subtle difference might have a large impact on the overall validity.

To overcome this, meta-certificates (in fact all EDA objects) are identified with their one-way "hash" value. There are several well-known and effective hash functions: EDA presently uses MD4 (developed by Ron Rivest, the co-inventor of RSA).

This hash function produces a string of 32 hexadecimal digits from any digital data.

Important properties of the hash function include:

- Because of the one-way nature of the hash, it is effectively impossible to construct an object with a hash matching a given value. I.e., it is impossible to create a "forged" object having the same hash as another.
- These properties allow the hash value of an object to be treated as its unique "fingerprint." If two objects have the same hash value, we can assume the two are the same — ***bit for bit.***

Therefore, in accepting a certificate, the user actually specifies (or verifies) a string of 32 hexadecimal digits. Users can accept any number of certificates. Acceptance is based on the user comparing the 32-digit number to some trusted source, such as a widely published listing. Once accepted, the user may then sign the hash so that it will be automatically recognized as accepted in the future.

By accepting a meta-certificate, a user demonstrates his trust that the associated meta-certifier will accurately identify organizations who are part of the EDA network, and constructs certificates for them in accordance with their wishes. Beyond this, the meta-certifier has no function.

If a certificate has co-signature requirements, the user is accepting the certificate's signatures only if the co-signature requirements are met.

Although most users will only need to accept meta-certificates, there are specialized reasons when it may be desirable to accept particular regular certificates.

Validating EDA Signatures and Certificates

Ultimate validity checking of a digital signature always lies with the recipient. It can be checked anytime, as many times, and by as many people as desired. Checking or displaying a signature in no way compromises any part of the system.

Although the following description omits substantial detail, it gives the flavor of how EDA signature proof-analysis proceeds:

Given a signed object and its proof packet (as constructed above) the entire signature and certificate structure is analyzed – toward the goal of deciding whether or not the object is ***acceptable***.

In the first step, all certificates and signatures are validated with RSA to ensure the contents are accurate, and unaltered. This includes verifying that each RSA signature(s) accurately reflects the data value of its object. Verification fails if there is a mismatch at any point, since such a mismatch would imply data damage (loss or tampering).

The unique ***hash*** of each certificate is also checked against the user's database to determine if it has been cancelled. (Cancellation is discussed later.)

In the next step, a reasonableness check, each signature is examined to ensure that the power it authorizes is in compliance with its certificate. Meta-certificates, which have no antecedent, are always presumed ratified.

Then, in an iterative process, an analysis is done to determine which other signatures and certificates are actually ratified according to the various respective certificate powers and rules; and when each ratification is scheduled to expire (based on the expiration dates of the certificates). (Although we are typically interested in whether a signature is presently ratified, in reviewing an archived document, we may want to verify that the signature was ratified when, say, the document was received, even through associated certificates may have expired in the interim.)

Finally, the user's Acceptance criterion (criteria) is applied selectively to ratified certificates. It is then percolated down through the hierarchy. Any ratified certificate which has been signed by an accepted certificate is also considered accepted.

The result of this validity checking process determines whether the primary object in question is signed by ratified and accepted certificates. If the object is signed by ratified and accepted certificates, the user may act on it as valid, and properly authorized. The acceptance process is completely "mechanical," and takes as input only the digital object, its proof packet, and the list of acceptance criteria.

If the object is only ratified but not accepted, then it cannot be accepted at face value. — It ***could*** be acceptable if the user were willing to enlarge his acceptance criteria (e.g., by accepting additional meta-certificates); however, this is not something to be done lightly. The determination of whether a (meta-)certificate is valid cannot be made simply by reviewing a certificate — it requires external knowledge and belief which must come from elsewhere.

If the object's authorization is not even ratified, then the EDA proof packet is inherently faulty or incomplete. This may be either because of tampering, or because various mandatory rules have not been entirely fulfilled.

Cancellation and Expiration Dates

From time to time, it will be necessary to cancel certificates before their natural expiration. This can occur for a number of reasons, including:

- Users cease to be affiliated with the organization which issued the authorization for their key. For example, employment is terminated.
- Users change position within an organization, requiring a reduction or alteration in authority.
- Users compromise their private key. This may be due to personal carelessness, or to penetration of local access-control security.

Under present EDA protocol, a certificate can be cancelled by:

- The user himself (for example, in the case he discovers he has accidentally compromised the key).
- The certificate creator (whose public key hash is embedded in the certificate), of the certificate being cancelled.
- Any direct ancestor certificate-creator of the certificate being cancelled (i.e., defined by recursive application of the previous rule).

Cancellation notices are special files, signed by an appropriate authority, specifying the certificate they cancelled, the reason for cancellation, the effective cancellation date, and the date the notice was issued. These notices must be made available to the population at large. Once received and ratified, only the hash of the cancelled certificate need be retained (and that, only until the expiration of the certificate). Any verification process should have access to the list of hashes.

Compatibility with X.500

EDA authorizing certificates can be treated as a superset of the X.500 directory certificates specifications. Since X.500 does not speak to the issue of generalized authority distribution, or co-signature capability, then these features are not applicable when using EDA certificates in "X.500 compatibility mode."

However, all EDA control objects – public and private keys, certificates, signatures, acceptance definitions, cancellation notices, and other miscellaneous EDA objects – are all designed as X.209 structures to allow maximum flexibility and compatibility with X.400 and X.500. (X.209 defines the "syntax" for transfer of information between X.400 applications.)

Analysis of the Potential EDA Weaknesses

EDA was designed to allow full use of digital signatures in actual business, in such a way that authorization could be ***automatically*** validated across a large and diverse population of enterprises.

Where EDA deviates most from conventional paper signatures is that the instrument for signing becomes the private key in a computer, rather than a pen. Since the private key is stored in encrypted form under a password invented by the user, inadvertent or unauthorized disclosure of this password becomes the weakest point in the system. Obviously, every user must be educated and encouraged to view their password as the signature to a "blank check," and to treat it accordingly.

However, the EDA concept of ***co-signature requirement substantially reduces this risk***. Any user who has significant EDA authority can be given ample co-signature requirements to reduce the risk as much as necessary. For example, if the risk that an arbitrary user's password is compromisable is, say, 1% (probably a large overestimate), then requiring 2 co-signatures reduces the risk to 1 in a million.

Of course, digital co-signatures also reduce the possibility of human corruption, for the same reason that paper co-signatures do. EDI without the safeguards of EDA could pose a much greater risk for economic crime and misuse than paper business. A cleverly insinuated digital file (without digital signatures, but which is taken at face value) leaves very little physical evidence — unlike forged paper instruments. In this area, EDA arguably ***affords stronger protection against white-collar fraud than paper signatures.***

Although there is still a need for the ability to cancel certificates, and distribute and maintain lists of cancellation notices, the urgency becomes less when "powerful" certificates are controlled with co-signature stipulations. As soon as co-signers are alerted, the risk of misuse becomes minimal. As mentioned earlier, this safeguard is also effective even if a certificate is compromised by an opponent without anyone else's knowledge.

Summary

Electronic Document Authorization is designed as a generalized technology for distribution of authority, the control of authority, and the validation of authority.

By defining co-signature requirements in conjunction with authorizations, EDA provides resilient security against corruptions and faults — in persons, computers, and their associated access control systems.

In this paper we discussed EDA primarily in relation to EDI. There are a large number of other applications where EDA's distributed authorization and automatic digital testing can be beneficially used (one example would be the use of multiple co-signatory guardians who must concur in order to grant access to computer data. This could be valuable where the data is kept in a single repository, and even more so if the data itself as well as the authorizers, were distributed).

EDA is a new kind of security — unlike many existing data security applications which rely strictly on access control, and where decisions must be rendered by one key individual at one focal point. EDA allows security to be distributed across many platforms, connected in arbitrary ways. Before allowing an action, EDA can force a consensus, based on flexible rules. Once a decision is reached, then whoever or whatever acts upon it, is assured that all appropriate rules were followed, and can even prove this to a third party if the need should arise. In particular, this has widespread application in business EDI, by regulating money authorizations among large organizations.

References

[1] Whitfield Diffie, "The First Ten Years of Public-Key Cryptography," *Proceedings of the IEEE*, Vol. 76, No. 5, pp. 560-577, May 1988.

[2] Whitfield Diffie and Martin E. Hellman, "Privacy and Authentication: An Introduction to Cryptography," *Proceedings of the IEEE*, Vol. 67, No. 3, pp. 397-427, March 1979.

[3] Whitfield Diffie and Martin E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, Vol. IT-22, No. 6, pp. 644-654, November 1976.

[4] R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Communications of the ACM*, Vol. 21, No. 2, pp. 120-126, February 1978.

The Place of Biometrics in A User Authentication Taxonomy

By Alex P. Conn, John H. Parodi, and Michael Taylor

Digital Equipment Corporation, 110 Spit Brook Road, Nashua NH, 03062

The characteristics of biometric authentication are discussed in the context of a taxonomy of authentication methods. The relative merits of passwords, smartcards, "see-through" authentication devices, and biometric authentication devices are described. Biometric authentication is not a panacea. It is a valid application of technology only if the physical security of the biometric reader is assured; biometric authentication is imperfect as a network-wide authentication scheme primarily because biometric characteristics are not secret.

Keywords: computer security, authentication, biometrics.

June 20, 1990

Introduction

In recent years we have seen an increasingly pervasive use of computers throughout the industrial and financial communities, the government, and our schools. With the increased use, there has been an increase in networking and dependence on the ability to move information reliably between systems. Unfortunately, this increased use of networks has also broadened the nature and scope of attacks that can be leveled at a computer system. While the nature and extent of threats is a rich and complex area of study, we can represent the threat simply as: (1) attacks against the user (e.g., stealing the credit card), and (2) attacks against the object (e.g., robbing the bank).

In this paper we concentrate on the user and how to ensure that identification and authentication are carried out properly. We present a taxonomy of various authentication techniques. Our thesis is that authentication techniques that work perfectly well in the local environment with stand-alone systems may be wholly inadequate to support authentication in a network. Our concern is that the current popularity of biometric schemes may allow them to be used in inappropriate ways.

In particular, we argue that biometrics is useful only as a local authentication technique *unless* assisted by other mechanisms such as encryption. That is, remote biometric authentication requires trust that: (1) the human is presently at the device that reads the biometric characteristic (2) the biometric reader itself is properly authenticated, and (3) the communication path between the reader and the authenticating system is of adequate integrity.

User Authentication Taxonomy

Authentication is the verification of a user's identity (to a given level of assurance). User authentication can be based on:

- What the user knows (e.g., a password)
- What the user has (e.g., a smartcard)
- What the user is (e.g., a biometric characteristic)

- Combinations of these (e.g., a smartcard that requires a PIN to be supplied)

The authentication techniques that correspond (more or less) to this taxonomy are passwords, smartcards and "see-through" authentication devices, and biometric devices, as described in the following sections.

Passwords

Passwords are the most common authentication mechanism and have several advantages:

- They are essentially free (no special hardware or equipment is needed).
- They are a familiar paradigm (they have been used on computers for years; PINs are used in conjunction with ATM cards).
- It is possible to make them relatively secure (e.g., using a one-time pad as the source for passwords[1]).
- They have been successfully used as part of encryption-based authentication schemes (they are used to "unlock" the key in the Kerberos system[2] [3]).

The disadvantages and limitations of passwords are:

- Passwords (excluding those taken from a one-time pad) do not provide strong authentication[1].
- Passwords are subject to eavesdropping on communication lines (not a problem if a one-time pad is the source for passwords).
- Passwords are vulnerable to an external dictionary attack unless software is designed to prevent the selection of easily-guessed passwords (this disadvantage disappears if a one-time pad is used as source for passwords). Unfortunately, bad passwords often result from the use of personal information about the user, e.g., spouse's name, date of birth, etc., and it is not clear how software could be designed to prevent the selection of such passwords.
- Passwords can be subject to internal attack as well. If cleartext passwords are stored on the host, they are subject to compromise in the event of a breakin or in the case of an untrustworthy user who is able to access the files that contain the passwords. Even where only hashed or encrypted passwords are stored, care must be taken to: (1) limit the bandwidth of brute-force attacks on passwords and thus reduce the vulnerability to attack (e.g., with some control over the number of retries before evasive action is taken, as in VMS), and (2) guard against the theft of the entire password file and a subsequent brute-force dictionary attack against the stolen copy.
- Passwords are often written down and thus their security is potentially limited by the physical security of the office. If users write down the password in a particularly bad place (e.g., writing down the PIN on the back of the ATM card), then compromise is even more likely.
- Passwords depend on schemes in which the host system must be trusted to "forget" the password the user supplies and terminate the authorization when requested (e.g., by a logout command). Thus, once a password is divulged, the user has no solid protection since the limit on the user's liability is entirely based on the trustworthiness of the host system.

Most of the above concerns can be addressed by enhanced password mechanisms. For example, the problem of broadcasting passwords on LANs can be addressed by encrypted connections, one time passwords, a Kerberos-like scheme and so forth. Dictionary attacks can be foiled by pass-phrase generators and a good pass-phrase generator might obviate or reduce the need to write down passwords as well.

See-Through Authentication

The see-through authentication approach may be characterized as a smartcard system that does not need a smartcard reader. In essence, the user acts as the conduit between the authentication device (often called a "see-through" card) and the computer. See-through authentication provides strong authentication of the

[1] CCITT X.509[4] describes the approach to strong authentication as "corroboration of identity by demonstrating possession of a secret key." We believe the most important aspects of strong authentication between two principals are that: (1) neither principal gains sufficient knowledge to subsequently impersonate the other and (2) observation of any or all authentication by a third party does not yield sufficient information to enable subsequent impersonation of either principal.

user to the system.[2] Two forms of see-through authentication have been widely discussed: (1) challenge/response, and (2) time-based. One example of the challenge/response approach is Polonius from Sytek, Inc. [5] .

The Polonius system uses cryptographic techniques to mediate a challenge/response protocol in which:

- The user establishes a connection with the host system (e.g., via a keyboard and display).
- In response to a prompt from the host, the user enters an ID.
- The host passes the user ID to an authentication server, which determines whether the ID is valid. If so, the authentication server passes a challenge and the proper response to the host (the challenge and response are computed using a key known to the authentication server and the user's see-through card).
- The host issues the challenge to the user and prompts for the response.
- The user enters a PIN and the challenge into the see-through card, which computes the proper response and displays it to the user.
- The user types in the value displayed by the see-through card and the host compares that value to the value the supplied by the authentication server, thus determining the authenticity of the user.

In some implementations, e.g., **WATCHWORD** from RACAL-GUARDATA Ltd. (an implementation of the Polonius scheme), [6] the same device can be set up so that it may be used to authenticate to more than one service, using the same PIN.

An example of a time-based see-through authentication scheme is the Access Control Encryption (ACE) system from Security Dynamics [7] . In the Security Dynamics product, there are two components, the Access Control Module (ACM), plugged into the computer, and the SecurID® card, carried by the user.

[2] In the Security Dynamics scheme, the Access Control Module can be configured to support strong authentication of the ACM to the user as well.

® SecurID is a registered trademark of Security Dynamics, Inc.

Each SecurID has an LCD that displays a pseudo-random number (PRN) at regular intervals. In addition, each user is provided with a PIN. At login, the user is asked to enter both the PIN and the PRN. (The PIN is associated with the serial number of the SecurID card.) Note that it is the device, not the user, that is authenticated to the system. There is an implicit assumption that the user is authenticated to the device (e.g., via the the PIN) and therefore that the device is correctly asserting that the user is present.

The card's generating algorithm is synchronized with the ACM. Thus the system "knows" that only the possessor of that card could provide that value (assuming the secrecy of the generating algorithm). The PIN, of course, also prevents use of the card in the event of loss or theft.

There are provisions for the use of an alternate "duress PIN." In addition, there is a provision for protecting the system from unauthorized attempts (the user may be asked to enter two valid PRNs in a row).

The Security Dynamics system also has a provision for authentication of the ACM to the user. In this configuration, the user first types the SecurID serial number, after which the ACM will display the pseudo-random number that is currently on the SecurID card. The user then enters the next PRN along with the PIN for authentication to the ACM.

The advantage of both see-through authentication schemes and smartcards is that strong authentication of the device is an intrinsic characteristic of the scheme. User authentication to the device involves: (1) possession of the device and (2) use of a PIN (typically). The disadvantages of both are: (1) the cost of the authentication devices, (2) the fact that it is necessary to carry the device, and (3) the user needs to remember the PIN. The particular advantage of see-through over smartcards is that no new hardware is required (i.e., you do not need a reader).

The obvious disadvantage to see-through authentication, as compared to smart cards, is that the user must enter some amount of information correctly in an exchange that may require greater care than a simple password entry. The problem can be compounded by noisy communication lines.

Another major disadvantage to see-through authentication, at least with current devices, is that it depends on symmetric encryption, which has several drawbacks:

- We know of no method that allows the use of symmetric keys for digital signatures without having to trust some kind of on-line notary or verification service. Such a service would have to store large numbers of secret keys and would therefore be a major target of attack. (If asymmetric keys are used, names and public keys can be paired and then encrypted off-line with verification service's public key. In this approach, only the public portion of the key must be available to software on the network; thus the server need not be trusted to protect the private portion of the key. See [8] for more information.)
- Symmetric encryption means that the authentication server possesses the encryption key; thus the authentication server must be trusted. Though replication of the server can prevent its being a single point of failure, it remains a single point of attack.
- Key distribution centers (for the management of symmetric keys) do not scale well for large networks.

While asymmetric (or public key) systems such as RSA solve these problems, it does not appear likely that asymmetric encryption can be used to add security value to a see-through authentication scheme. Public-key algorithms generate large blocks of information, therefore a challenge or response must be hundreds of bits in length. To enter a challenge of, for example, 512 bits, a user would have to type 128 hexadecimal digits, which results in an unbearably cumbersome user interface.

Smartcards

Smartcards provide strong authentication as well as solutions to all the disadvantages mentioned in the Passwords section. While not an authentication issue per se, an additional advantage of a smartcard based on public key cryptosystems is the ability to digitally sign documents.[9]

In addition, public-key smartcards allow an architecture in which no principal is given the means to impersonate another principal, nor are private or secret keys stored in an online server that, if compromised, could provide the means for impersonation.[8]

The main disadvantages of smartcards are:

- Smartcard readers must be integrated into new workstations, PCs and terminals, and integration with a significant set of existing equipment will be necessary as well.

  The smartcard reader will be a potential point of attack as long as it must provide some of the "smarts" (e.g., PIN entry, display, etc.). At the time of this writing, smart cards with keypads, LCD displays, and the computing power necessary for digital signatures are not widely available.
- Each user must be given a smartcard at a potentially substantial aggregate cost.
- Some scheme is needed in the event that a smartcard is forgotten, lost, or stolen.
- Some scheme (e.g., PIN codes) is needed to prevent use by others and to ensure that the user is authenticated to the device. Without such a scheme, the smartcard can only authenticate itself and cannot validly assert the presence of a particular user. The management of any such scheme incurs some cost.

More details about authentication based on asymmetric encryption can be found in [10] and [11] .

Biometric Devices

The use of biometric devices rather than smartcards has been proposed to address many of the issues mentioned above. Biometric devices read some physical characteristic of the user and can be categorized as relying on either: (1) passive characteristics such as fingerprint, the pattern of blood vessels on the retina, etc., or (2) active characteristics such as handwritten signature or voice characteristics. Biometric authentication uses some form of pattern recognition to determine whether the biometric characteristics presented are considered equivalent (within some threshold or tolerance) to the stored values for that individual.

The advantages of biometric devices are:

- The system is easy to use. Although a biometrics system requires some training and perhaps the use of a PIN, the essence of the system is that a machine reads some characteristic from a passive user (e.g., retina or finger prints) or a from a user's action (e.g., a manual signature).

- The user does not need to carry a token that could be forgotten, lost, or stolen. Biometric characteristics can potentially provide the strongest binding yet known between the user and the authentication information. This makes a biometrics scheme very attractive in certain applications, e.g., entry to a building.

- If used to control entry to a physically secure area, there is essentially no additional cost per user, once the biometric reader is purchased (except for storage of biometric characteristics). However, the cost of biometric readers rises quickly if they must be installed at each point of login to a computer system.

We argue that biometrics, by itself, is unsuitable for any application outside the bounds of local authentication for the following reasons. If biometrics is to be used for remote rather than local authentication, the design must protect against two distinct kinds of attack:

- Spoofing the biometric reading mechanism itself (e.g., providing the thumbprint without the thumb's owner being present or using a high quality voice generator to fool voice recognition circuitry)

- Bypassing the biometric reader entirely

These threats are specific to biometric schemes because they are based on the fact that biometric characteristics are not secrets and must not be thought of as secrets. (It is absurd, for example, to think that one can protect one's thumbprint from disclosure.) Even for more "sophisticated" biometric characteristics such as retina prints or handwritten signature analysis, the average person could not protect those characteristics from being "read" or captured by a malicious party.

The threat of spoofing is made possible by the fact that once biometric characteristics are known, somebody can design devices capable of supplying those characteristic to within the tolerance of the reader. The only defenses against such an attack are to: (1) guard the reader, thus ruling out obvious biometric bypassing equipment or duress (discussed below), and (2) use biometric readers that are very difficult to spoof. An organization's policy toward such readers must balance the cost of a reader that has the necessary level of discrimination (including guards, if appropriate) against the value of the resource being protected.

The threat of bypassing the reader does not apply to passwords, see-through devices, or smartcards because in those cases, the reader is only a conduit for a secret. If properly implemented, possessing or replacing the password reader (the keyboard) or the smartcard reader should never provide a means for obtaining the secret.

For remote biometric authentication, however, bypassing the reader is a serious threat. Since biometric characteristics are not secret, mere possession of the bits that correspond to an individual proves nothing. The authentication value comes from the knowledge that the bits are coming directly from a valid reader that is known to be securely connected to the machine that uses the bits for authentication. Without adequate protection, an attacker could: (1) physically replace the reader with one that emits the characteristics of a specific target individual or (2) compromise the authentication at the host to which the reader is connected or via nodes elsewhere on the network (e.g., by a replay attack).

To protect against such attacks, some means must be used to place the biometric reader within the same security perimeter as the remote node to which the user is authenticating. That is, a "secure connection" is needed between the reader and the remote node that guarantees both the authenticity of the reader itself and of the bits it sends authenticating the user. In most cases, the most practical method of affording protection from these threats involve encryption and timestamps. Thus to guarantee both the authenticity of the reader and the integrity of any information it transmits, we argue that the reader must be able to employ encryption (either encrypt the entire message or digitally sign some kind of message digest).

Using encryption to protect the biometric authentication protocol appears to result in a particularly thorny problem. Since biometric authentication does not (indeed cannot) require that the read characteristics exactly match the stored characteristics, it would seem that an authenticating server must have the unencrypted biometric available. I.e., the server must either have the "cleartext" biometric bits or the decryption key available—and in the event that the server is compromised, these are equivalent (this is true for both symmetric and asymmetric encryption schemes). The reason for this requirement is that a "variation threshold algorithm," which allows authentication for a close (but not perfect) match between the stored and read values, would not be able to deal with a comparison of two "close" values in their encrypted form.[12]

To avoid the above problem, it is conceivable that the reader might carry out the biometric algorithms and simply send an "accept" or "deny" message to the remote system. However, any requirement for carrying out the actual biometric authentication locally rather than simply shipping the bits read could significantly increase the cost of the biometric reader. The local node could help in the processing, but the node would then also have to be trusted by the remote system. Note that in any of these implementations, the message from the reader (or local node) must also be protected by encryption. Thus for secure remote biometric authentication, the cost of encryption must be added to the processing costs associated with the biometric authentication. In fact, any remote authentication scheme must incur the overhead cost (usually encryption) associated with authenticating the device (see-through device, smart card, or biometric reader).

The following list describes other disadvantages to biometric authentication:

- The readers are relatively costly—currently at least an order of magnitude more than the simplest smartcard reader (to achieve a reasonable level of correct biometric authentications).
- In the event of compromise, changing biometric characteristics is essentially impossible.
- With injuries (e.g., a cut on the finger or a sprained wrist), it may become difficult to authenticate.
- The scheme is not readily adapted to other uses such as digital signatures (as is also true of passwords, of course).
- A potentially significant amount of computation is required to verify the biometric bits to the threshold needed to allow for a close (but not perfect) match between the stored and read values in order to ensure that only the right individual is "passed." (While asymmetric encryption also requires a significant amount of computation, we have shown that for remote biometric authentication, encryption is required anyway; thus the biometric verification is additional overhead.)
- Some biometric devices, such as those reading fingerprints, could be defeated using available techniques for faking fingerprints. Since biometric information is not secret, it can be argued that given enough incentive, it is only a matter of time before someone builds a device that defeats a given biometric authentication scheme. Whatever the biometric pattern chosen as the "authenticator," that pattern might either be obtained (e.g., fingerprints from a bar glass) or fabricated as technology evolves.
- While all authentication methods can be subverted by coercion of the user, certain biometric approaches appear to be even more vulnerable than most methods. E.g., an unconscious user cannot be made to divulge a password or PIN but a fingerprint or retina print could be obtained. (The following section touches on the interesting topic of authentication under duress.)

Active biometric systems (as opposed to static characteristics like fingerprints or retina prints) are more difficult to defeat. For example, it is possible to identify a user based on the analysis of typing patterns. However, typing pattern analysis might have problems in commercial systems (especially), where complex operations are reduced to invocation by a very few keystrokes or even a point-and-click interface and typing patterns become more difficult to discern.

Handwritten signature verification, voice recognition, and typing analysis all need to be able to deal with foreseeable changes to the biometric characteristics. E.g., a broken arm or even a sprained finger could make authentication difficult for the manual signature or keystroke analysis approaches. A head cold or dental work might cause problems for a voice recognition system.

Authentication Under Duress

Some authentication systems incorporate the idea of having available two different PINs, one for normal use and one to be used when under duress. The issues are:

1. Does authentication require only a passive user role (could a criminal accomplish authentication with a drugged or unconscious victim, possibly without the victim ever knowing that authentication information had been obtained by the criminal)?
2. Will stress (likely to be present with duress) make some biometric schemes (e.g., voice recognition and signatures) impossible to use under duress?
3. Is there some means for appearing to correctly authenticate while really warning the system of the duress condition (e.g., Polonius)? If so, how many users would really be willing (or remember how) to use the duress warning with a gun pointed at their head?

Any biometric scheme for which the answers to 1 and 2 could be "yes" is potentially less effective than other authentication choices. Note that if the answer to 2 is "yes," the authentication scheme is still effective from the system's point of view; i.e., it is fail-safe. However, in a duress situation, the *user's* security may be in jeopardy. In fact the larger question of whether the systems should work when the user is under any form of stress must still be examined. This is essentially a matter of security policy. All authentication schemes we have examined except for biometrics could easily be adapted to allow for alternate PINs or passwords to be used as a duress warning. For biometrics, it would be necessary to add some kind of PIN mechanism, which reduces the simplicity of biometrics.

The question of notifying the system about a duress situation is clouded by the knowledge that if a gun is pointed at one's head, there is a good chance that one will hand over password, smartcard (with the "real" PIN), or anything else the gun wielder wants.

Conclusions

The disadvantages to biometric authentication are rooted in the fact that biometric characteristics are not secret. Because they are not secret, a biometric characteristic by itself cannot be unforgeable proof that the user is at a particular remote node. Confidence in the presence of the user is based only on trust in the node (or biometric reader) that makes the assertion. Thus, the disadvantages to biometrics become apparent only in the context of a computer network, in which a user might want to authenticate to a remote node.

In applications where no remote authentication is contemplated and physical security is assured (e.g., entry to a building or entry to a computer room that is guarded 24 hours a day), biometric authentication is a valid application and could be a very attractive option because of its potential ease-of-use characteristics.

It has been suggested that biometrics (rather than a PIN) would be a good way for a user to authenticate to a smartcard. If the biometrics approach uses "static" characteristics, the advantages when balanced against the threat of a serious attack are dubious. If someone is willing to drug or knock the user unconscious, that user's biometric characteristics are much more vulnerable than a password or PIN. On the other hand, a dynamic biometric, such as handwriting analysis, might be reasonable.

If the probability of attack on the smartcard itself is low, then biometric authentication to the smartcard might be considered. The advantage is that such an approach protects against mild cases of incompetence, e.g., users who share a PIN or who might scratch the PIN into the casing of the smartcard.

In either case, in order to be secure, the biometric reader would have to be "local" to the card; that is, either actually on the card or directly connected to the card. If the biometric reader is connected to the smartcard via the host, the reader is now "remote" in the sense that, without protection, the host could compromise the authentication exchange. Without that protection, the scheme is more expensive and less secure than a smartcard that uses a PIN.

In the context of a computer network, the idea of a permanent compromise of one's biometric characteristics is frightening. The inability to use the device for digital signatures is also a serious drawback. The cost for a "good enough" implementation to resist spoofing could be high. The major advantages of biometric devices are: (1) ease of use (potentially), (2) low additional cost per user, and (3) no problem with loss or theft.

We believe that loss or theft of smartcards could be dealt with using a reasonable temporary card process administered by security personnel, coupled with a PIN code. While theft is not an issue with biometric devices, injury could have the same impact, at least temporarily. When smartcards can be obtained for $10, their advantages are likely to override other cost considerations, at least in the area of distributed authentication.

References

[1] Butler Lampson, of Digital Equipment Corporation, in a private communication.

[2] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer, "Project Athena Technical Plan, Section E.2.1, Kerberos Authentication and Authorization System," ©1985, 1986, 1987 by the Massachusetts Institute of Technology

[3] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller, "Kerberos: An Authentication Service for Open Network Systems," Presented at Winter USENIX 1988, Dallas, TX

[4] CCITT IXth Plenary Assembly CCITT, Melbourne, 1988, Document 47, Study Group VII, Report R 38, Recommendation X.509

[5] Raymond M. Wong, Thomas A. Berson, and Richard J. Feiertag, "Polonius: An Identity Authentication System," CH2150-1/85/0000/0101 © 1985 IEEE.

[6] **WATCH**WORD Generator User's Manual, RACAL-GUARDATA LIMITED, Richmond Court, 309 Fleet Road, Fleet, Hampshire, England GU138BU

[7] Final Evaluation Report of Security Dynamics Access Control Encryption System, National Computer Security Center, 9800 Savage Road, Fort George G. Meade, MD 20755-6000, CSC-EPL-87/001 Library No. S228,455

[8] M. Gasser, A. Goldstein, C. Kaufman, B. Lampson, "The Digital Distributed System Security Architecture", Proceedings, 12th National Computer Security Conference, Baltimore, MD, Oct. 1989, pp. 305-319.

[9] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," Communications of the ACM 21(2), 120–26, 1978.

[10] John Linn, "Practical Authentication for Distributed Computing," Digital Equipment Corporation, Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy, May 1990, Oakland CA, pp. 31-40.

[11] Morrie Gasser and E. McDermott, "An Architecture for Practical Delegation in a Distributed System," Digital Equipment Corporation, Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy, May 1990, Oakland CA, pp. 31-40.

[12] John Linn, of Digital Equipment Corporation, from an informal discussion.

Acknowledgments

We are indebted to Morrie Gasser, George Gajnak, Charlie Kaufman, John Linn, Steve Lipner, Charles Lo, Colin Rous, Bill Shockley, and Earl Van Horn for their insightful comments on drafts of this paper.

Suggested Reading

"Building A Secure Computer System," Morrie Gasser, © 1988 Van Nostrand Reinhold, 115 Fifth Avenue, New York, New York, 10003.

"VMS Guide to System Security," © June 1989 by Digital Equipment Corporation, Order number AA-LA40B-TE, Digital Equipment Corporation, Maynard Massachusetts.

NON-FORGEABLE PERSONAL IDENTIFICATION SYSTEM USING CRYPTOGRAPHY AND BIOMETRICS

Glenn Rinkenberger and Ron Chandos, Motorola Government Electronics Group, 8201 E. McDowell Rd. Mail-Stop H1102, Scottsdale, AZ 85252

ABSTRACT

This paper describes a concept for combining cryptographic and biometric techniques to provide an unforgeable set of authentication credentials absolutely linked to only the rightful owner. These credentials can then be presented at a remote site, and provide convincing proof that the presenter is who he claims to be and that he holds the privileges he claims to hold. A fully operational feasibility model, based on facial image and fingerprint biometrics, is described. Also discussed is a method for adapting the concept to validate users of the STU-III secure telephone, and a multi-user computer network.

PROBLEM

Modern societies often experience the problem of positive identification of a single individual and determining privileges associated with that individual. In the government realm, the problem of personal authentication is closely coupled to security issues involving physical access control, obtaining classified material, visiting off-site facilities, and logging onto classified multi-user computers or networks. Within the public domain, the problem is most evident during everyday financial transactions such as the use of credit cards, check cashing, and automatic tellers.

In both the government and public domain, there exists a strong need for personal authentication. The authentication process enables a person requesting a service or privilege to prove positively that he is entitled to that privilege or service. An ideal authentication system provides convincing proof that an individual is who he claims to be, and that he is entitled to the privileges he claims to have.

The most pervasive systems in use today are exemplified by the credit card application. In this application, privileges are identified by the type of card, and the requester identified by having possession of the card. Many retail sites also employ card readers linked via modems over phone lines to access a central computer base to verify card validity. Note that this procedure validates the card, with no regard for whether the bearer of the card is the rightful owner. Consequentially, these types of systems offer limited security, and are defeated or compromised when the credential is modified, lost, stolen, or forged.

In addition to possessing the card, systems like the automatic tellers used by the banking industry require a second authentication step in the form of a password or identification number, presumably known only by the valid holder of the card.

The issuance of personal identification in the government sector is particularly complex due to the desire for compartmentalization of access to classified data and facilities. Another complication is created by the lack of a central authority to control identification methods and policy. The systems in use by various government agencies are generally different and non-interoperable, causing inconvenience, delays, and extra procedures and paperwork when inter-agency transactions are required.

Motorola has developed a concept for combining cryptograpy and biometrics to provide an unforgeable set of authentication credentials absolutely linked to only the rightful owner. These credentials can then be presented at a remote site to provide convincing proof that the presenter is who he claims to be and that he holds the privileges he claims to hold.

THE NEW CONCEPT

The motivation for the proposed authentication system is based on severe shortcomings of identification systems in common usage today. All of today's systems appear deficient in one or more of the following areas.

- Ease in forging the identification credentials-
- Lack of positive authentication tied to a physical person-
- Vulnerability due to lost, stolen, or forged credentials-
- On-line linkage to a central data base-

The new concept provides a biometric and cryptographic basis for proving that the bearer of the credentials is the individual to whom they were issued, and that the attributes or privileges conveyed by the credentials were certified and bound to the individual.

The new approach, shown in Figure 1, involves a trusted credential issuing agency (Authorization Segment) and numerous transaction sites (Validation Segments). The Authorization Segment is responsible for validating the identity, attributes, and privileges for an individual requesting credentials. When validated, the credential media is generated and given to the requestor. The credential media can then be tendered at any of the Validation Segment sites where the holder desires to complete a transaction. The Validation Segment equipment then processes the information contained in the credentials and determines whether the presenter should be allowed to complete the desired transaction.

Specific details describing this concept follow the background information presented below. The recommended system is based on three proven technologies, biometrics, public key cryptography, and memory cards.

BIOMETRICS-

The biometric contribution allows basing the identification decision on some immutable trait unique to the specific individual. Commonly

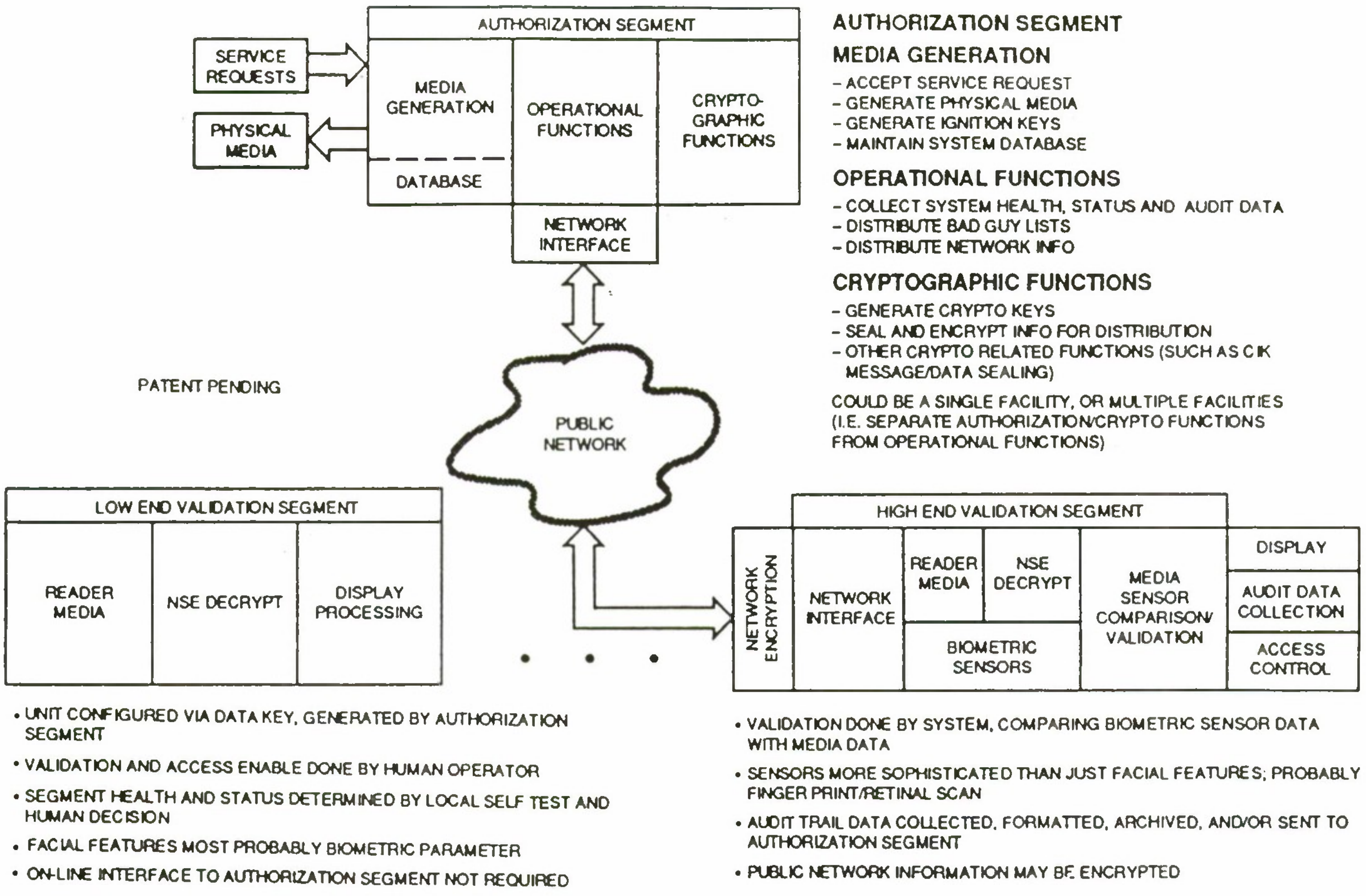

AUTHORIZATION SEGMENT
SERVICE REQUESTS
PHYSICAL MEDIA
MEDIA GENERATION
DATABASE
OPERATIONAL FUNCTIONS
CRYPTO-GRAPHIC FUNCTIONS
NETWORK INTERFACE
AUTHORIZATION SEGMENT
MEDIA GENERATION
- ACCEPT SERVICE REQUEST
- GENERATE PHYSICAL MEDIA
- GENERATE IGNITION KEYS
- MAINTAIN SYSTEM DATABASE
OPERATIONAL FUNCTIONS
- COLLECT SYSTEM HEALTH, STATUS AND AUDIT DATA
- DISTRIBUTE BAD GUY LISTS
- DISTRIBUTE NETWORK INFO
CRYPTOGRAPHIC FUNCTIONS
- GENERATE CRYPTO KEYS
- SEAL AND ENCRYPT INFO FOR DISTRIBUTION
- OTHER CRYPTO RELATED FUNCTIONS (SUCH AS CIK MESSAGE/DATA SEALING)
COULD BE A SINGLE FACILITY, OR MULTIPLE FACILITIES (I.E. SEPARATE AUTHORIZATION/CRYPTO FUNCTIONS FROM OPERATIONAL FUNCTIONS)
PATENT PENDING
PUBLIC NETWORK
LOW END VALIDATION SEGMENT
READER MEDIA
NSE DECRYPT
DISPLAY PROCESSING
HIGH END VALIDATION SEGMENT
NETWORK ENCRYPTION
NETWORK INTERFACE
READER MEDIA
NSE DECRYPT
BIOMETRIC SENSORS
MEDIA SENSOR COMPARISON/ VALIDATION
DISPLAY
AUDIT DATA COLLECTION
ACCESS CONTROL
• UNIT CONFIGURED VIA DATA KEY, GENERATED BY AUTHORIZATION SEGMENT
• VALIDATION AND ACCESS ENABLE DONE BY HUMAN OPERATOR
• SEGMENT HEALTH AND STATUS DETERMINED BY LOCAL SELF TEST AND HUMAN DECISION
• FACIAL FEATURES MOST PROBABLY BIOMETRIC PARAMETER
• ON-LINE INTERFACE TO AUTHORIZATION SEGMENT NOT REQUIRED
• VALIDATION DONE BY SYSTEM, COMPARING BIOMETRIC SENSOR DATA WITH MEDIA DATA
• SENSORS MORE SOPHISTICATED THAN JUST FACIAL FEATURES; PROBABLY FINGER PRINT/RETINAL SCAN
• AUDIT TRAIL DATA COLLECTED, FORMATTED, ARCHIVED, AND/OR SENT TO AUTHORIZATION SEGMENT
• PUBLIC NETWORK INFORMATION MAY BE ENCRYPTED

Figure 1- System Concept

used biometric techniques include facial features, fingerprints, voice prints, retinal scans, static and dynamic handwriting characteristics, and hand geometries.

All of these biometric traits are currently able to be digitized and stored in a 'reasonably' sized data base (reasonableness defined in terms of the capacity of existing and proven memory cards, with allowances for other data, described later). The resulting biometric data base is uniquely linked to one specific individual. Table 1 is a brief and somewhat qualitative survey of some of the currently available published biometric industry data[1]. Since testing, decision thresholds, and reporting methods differ widely between vendors, this data should be viewed in a conceptual rather than a comparative manner.

Table 1- Biometric Industry Survey Information

Biometric Template	False Reject Rate	False Accept Rate	Size
Fingerprint	2%	0	10 Kbits
Hand Geometry	1%	0.4%	1 Kbits
Retinal Scan	3%	0	1 Kbits
Voiceprint	4%	0.5%	10 Kbits
Dynamic Signature	1%	0	1 Kbits

PUBLIC KEY CRYPTOGRAPHY-

The concept of public key cryptography provides several benefits to the proposed personal authentication concept. The important property of the public key cryptography is the separate and distinct encrypt and decrypt keys, where one element of the key pair can be made widely available without providing information related to the other half of the key pair. In the typical public key communications scenario, the encrypt key is made available to the public, while the decrypt key is tightly guarded by the owner. This allows anyone to send secure information to the owner, with the owner being the only party able to read the data message.

For the authentication system, the cryptographic key pair usage is opposite from the communications scenario. That is, the encrypt key is held privately and the decrypt key is distributed. The credential issuing agency is the sole possessor of the encrypt key, while distributing the decryption key to all personal identification sites.
The resulting cryptographic benefit is two-fold. First, the threat of forged credentials is removed, since forgery is impossible without knowledge of the encryption key (held and protected by the trusted issuing agency). Next, the transaction site, upon decrypting and validating the presented credentials, can safely conclude that the credentials were indeed generated by the trusted issuing agency and can therefore trust all information on the credentials.

MEMORY CARD TECHNOLOGY

The memory card technology is rapidly advancing, with several million memory cards in use in Europe and Japan. The cards provide a conveniently portable medium, allowing the holder to transport large quantities (hundreds of kilobytes) of digital information in a credit card sized unit. The proposed system uses the credential to contain the encrypted biometric trait information for the proper holder, and numerous encrypted data files indicating attributes and/or privileges validly held by the holder.

Although these three technologies are individually mature and well understood, the authentication system uses them in a unique combination, allowing unforgeable proof that the individual and his claimed privileges are valid.

AUTHORIZATION SEGMENT

The authorization segment (Figure 2) will be one or at most a limited number of sites that produce the credentials. The authorization segment must first either generate or receive from some other source properly certifiable information about the individual for which credentials are to be prepared. Existing methods presently used to grant security clearances or credit cards are examples of possible certification methods.

The biometric(s) used for identification are application dependent and are influenced by the required security, the degree of human involvement at the transaction site, and human factors (inconveniences) tolerable by the presenter. For example, facial images are applicable to manned sites where a guard is available to make a match/no-match decision. This biometric is very unobtrusive from the presenter's point of view. For unmanned sites, biometrics such as fingerprints or retinal scans are more amenable to machine-based match/mismatched decisions. These biometrics are somewhat more inconvenient to the presenter. Note that the credential may contain several biometric files, allowing combining the traits for very high security applications, or using the biometrics individually for several applications with differing security levels.

Once the trait method(s) is selected, it is necessary to gather the trait data. This data may be collected directly on site from the individual or may be communicated to the site via mail or electronic means. Fundamental to the process is the conversion of this trait data to digital data in a fixed format. Existing commercial equipments are available which perform this operation.

After the trait data has been collected and formatted, it may now be appended with additional identification information such as name, social security number, etc. This data set may be further augmented by additional non-identification information, representing any information the authorizing agency wishes to include as part of the credentials. Examples include organizational levels, clearance levels, access pin codes, foreign travel allowances, special compartment accesses, etc. Once completed, the data set and

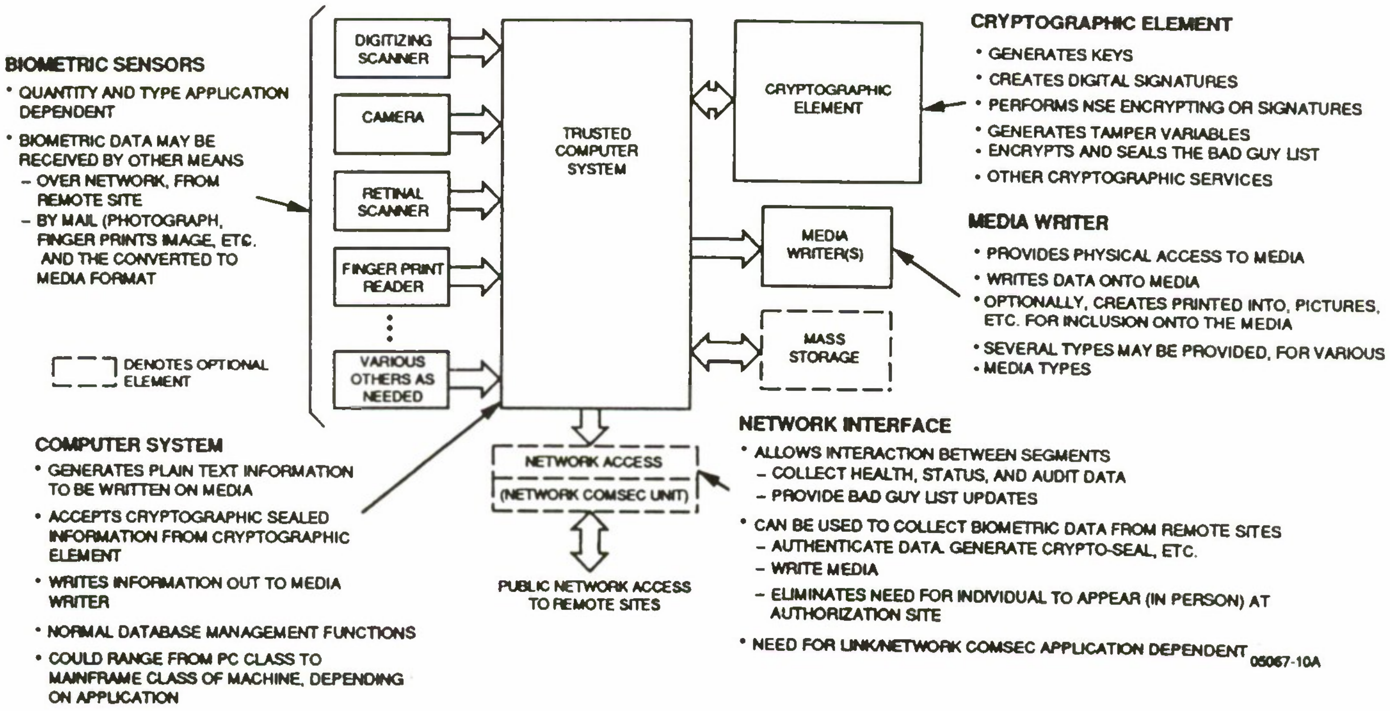

BIOMETRIC SENSORS
• QUANTITY AND TYPE APPLICATION DEPENDENT
• BIOMETRIC DATA MAY BE RECEIVED BY OTHER MEANS
– OVER NETWORK, FROM REMOTE SITE
– BY MAIL (PHOTOGRAPH, FINGER PRINTS IMAGE, ETC. AND THE CONVERTED TO MEDIA FORMAT
DENOTES OPTIONAL ELEMENT
DIGITIZING SCANNER
CAMERA
RETINAL SCANNER
FINGER PRINT READER
VARIOUS OTHERS AS NEEDED
TRUSTED COMPUTER SYSTEM
CRYPTOGRAPHIC ELEMENT
MEDIA WRITER(S)
MASS STORAGE
NETWORK ACCESS
(NETWORK COMSEC UNIT)
PUBLIC NETWORK ACCESS TO REMOTE SITES
CRYPTOGRAPHIC ELEMENT
• GENERATES KEYS
• CREATES DIGITAL SIGNATURES
• PERFORMS NSE ENCRYPTING OR SIGNATURES
• GENERATES TAMPER VARIABLES
• ENCRYPTS AND SEALS THE BAD GUY LIST
• OTHER CRYPTOGRAPHIC SERVICES
MEDIA WRITER
• PROVIDES PHYSICAL ACCESS TO MEDIA
• WRITES DATA ONTO MEDIA
• OPTIONALLY, CREATES PRINTED INTO, PICTURES, ETC. FOR INCLUSION ONTO THE MEDIA
• SEVERAL TYPES MAY BE PROVIDED, FOR VARIOUS
• MEDIA TYPES
NETWORK INTERFACE
• ALLOWS INTERACTION BETWEEN SEGMENTS
– COLLECT HEALTH, STATUS, AND AUDIT DATA
– PROVIDE BAD GUY LIST UPDATES
• CAN BE USED TO COLLECT BIOMETRIC DATA FROM REMOTE SITES
– AUTHENTICATE DATA, GENERATE CRYPTO-SEAL, ETC.
– WRITE MEDIA
– ELIMINATES NEED FOR INDIVIDUAL TO APPEAR (IN PERSON) AT AUTHORIZATION SITE
• NEED FOR LINK/NETWORK COMSEC APPLICATION DEPENDENT
COMPUTER SYSTEM
• GENERATES PLAIN TEXT INFORMATION TO BE WRITTEN ON MEDIA
• ACCEPTS CRYPTOGRAPHIC SEALED INFORMATION FROM CRYPTOGRAPHIC ELEMENT
• WRITES INFORMATION OUT TO MEDIA WRITER
• NORMAL DATABASE MANAGEMENT FUNCTIONS
• COULD RANGE FROM PC CLASS TO MAINFRAME CLASS OF MACHINE, DEPENDING ON APPLICATION
05067-10A

Figure 2- Authorization Segment

biometric information is passed to an encryption function. Note that multiple levels of public key encryption can be applied to various portions of the attribute or privilege data base. That is, the trait data and general portion of the data base (example, name) can be encrypted in one key. Additional portions (example, special privileges) can be encrypted on a second key. The second decrypt key is only available to a subset of the validation segment, such as a site where special privileges are needed and recognized. In this way, the general validation site is unaware that the card holder possesses any special privileges.

The encrypted data represents the unforgeable credential data for a given individual. This data may be written to a suitable digital storage medium to be used by the individual as his personal identification and attribute or privilege credentials. Many forms of the medium, such as a credit card, may also contain the commonly used printed information on the medium, as well as the encrypted biometric and attribute electronic data. The printed information and pictures makes the credentials look like the traditional badge or driver's license ID, allowing it to be used in a non-electronic manner for low security transactions.

VERIFICATION SEGMENT

The Verification Segment (Figure 4) consists of one or a multiplicity of sites which provide authentication or access control functions based on the presentation of credentials. The nature of the verification site will vary considerably based on the type of traits used for the identification process. The simplest case is a manned site where a facial photograph is used as the identification trait. The presenter would provide his credentials to a reading device which reads the digital data from the medium and performs the decryption function. The more complex sites would include biometric sensors and automatic authentication comparison software. Figure 5 is a typical description of the processing performed at the verification site.

In many applications the verification site will provide a "log" or audit trail function. This function would replace the current sign-in procedure and provide a recorded history, most likely via hard disk and diskette storage. The audit trail information could also be written onto the credential providing a personal record of all the places where the credential had been used.

PROOF OF CONCEPT SYSTEM

This research effort has resulted in the development of a proof of concept hardware/software system using the facial photograph and the finger print as the identification traits. Figure 6 illustrates the hardware structure. The system is capable of capturing both the front and profile pictures of the individual as well as one or more finger prints. The biometric data is augmented with text information, encrypted and written onto a memory card. The text information contains a complete drivers license and passport as well as security

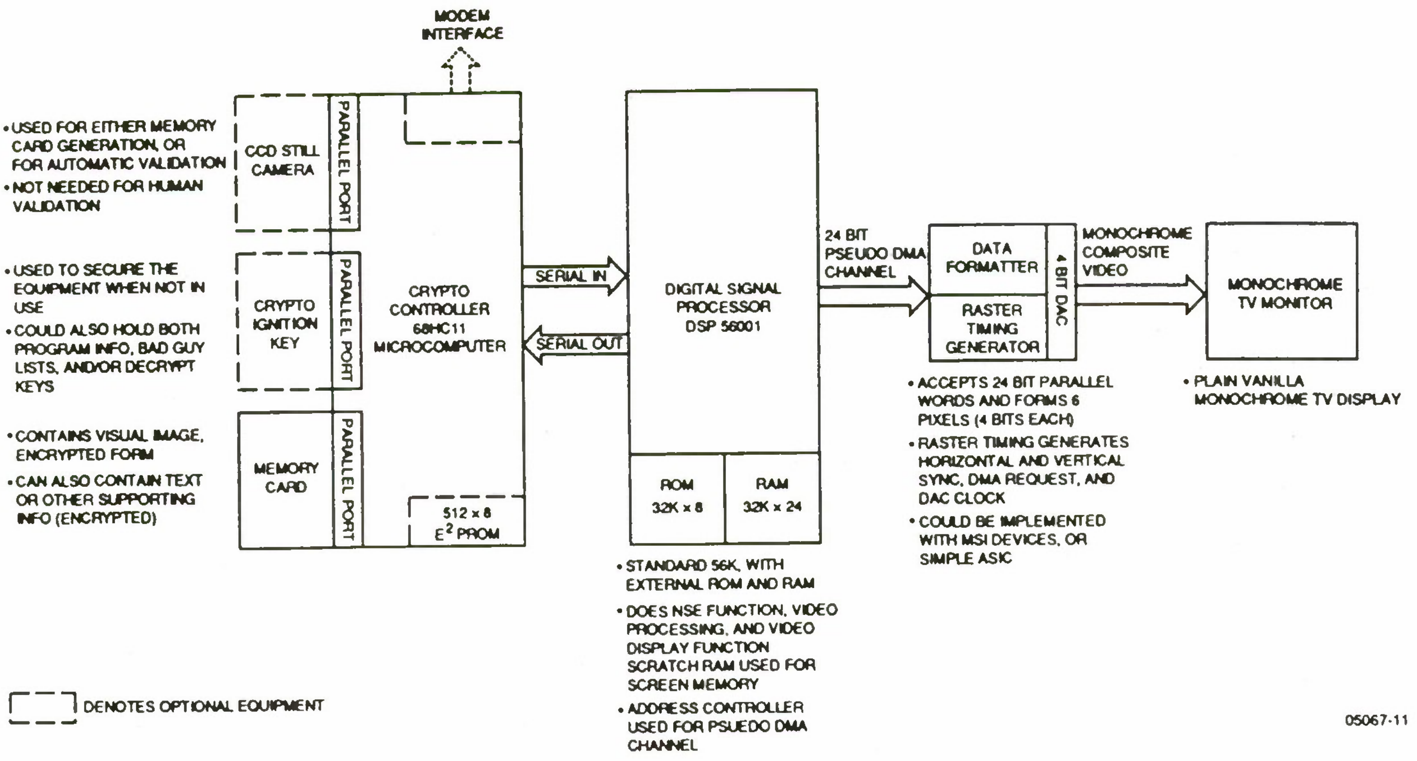

MODEM INTERFACE
• USED FOR EITHER MEMORY CARD GENERATION, OR FOR AUTOMATIC VALIDATION
• NOT NEEDED FOR HUMAN VALIDATION
CCD STILL CAMERA
PARALLEL PORT
• USED TO SECURE THE EQUIPMENT WHEN NOT IN USE
• COULD ALSO HOLD BOTH PROGRAM INFO, BAD GUY LISTS, AND/OR DECRYPT KEYS
CRYPTO IGNITION KEY
• CONTAINS VISUAL IMAGE, ENCRYPTED FORM
• CAN ALSO CONTAIN TEXT OR OTHER SUPPORTING INFO (ENCRYPTED)
MEMORY CARD
CRYPTO CONTROLLER 68HC11 MICROCOMPUTER
512 x 8 E² PROM
SERIAL IN
SERIAL OUT
DIGITAL SIGNAL PROCESSOR DSP 56001
ROM 32K x 8
RAM 32K x 24
• STANDARD 56K, WITH EXTERNAL ROM AND RAM
• DOES NSE FUNCTION, VIDEO PROCESSING, AND VIDEO DISPLAY FUNCTION SCRATCH RAM USED FOR SCREEN MEMORY
• ADDRESS CONTROLLER USED FOR PSUEDO DMA CHANNEL
24 BIT PSEUDO DMA CHANNEL
DATA FORMATTER
RASTER TIMING GENERATOR
4 BIT DAC
• ACCEPTS 24 BIT PARALLEL WORDS AND FORMS 6 PIXELS (4 BITS EACH)
• RASTER TIMING GENERATES HORIZONTAL AND VERTICAL SYNC, DMA REQUEST, AND DAC CLOCK
• COULD BE IMPLEMENTED WITH MSI DEVICES, OR SIMPLE ASIC
MONOCHROME COMPOSITE VIDEO
MONOCHROME TV MONITOR
• PLAIN VANILLA MONOCHROME TV DISPLAY
DENOTES OPTIONAL EQUIPMENT
05067-11

Figure 4- Verification Segment

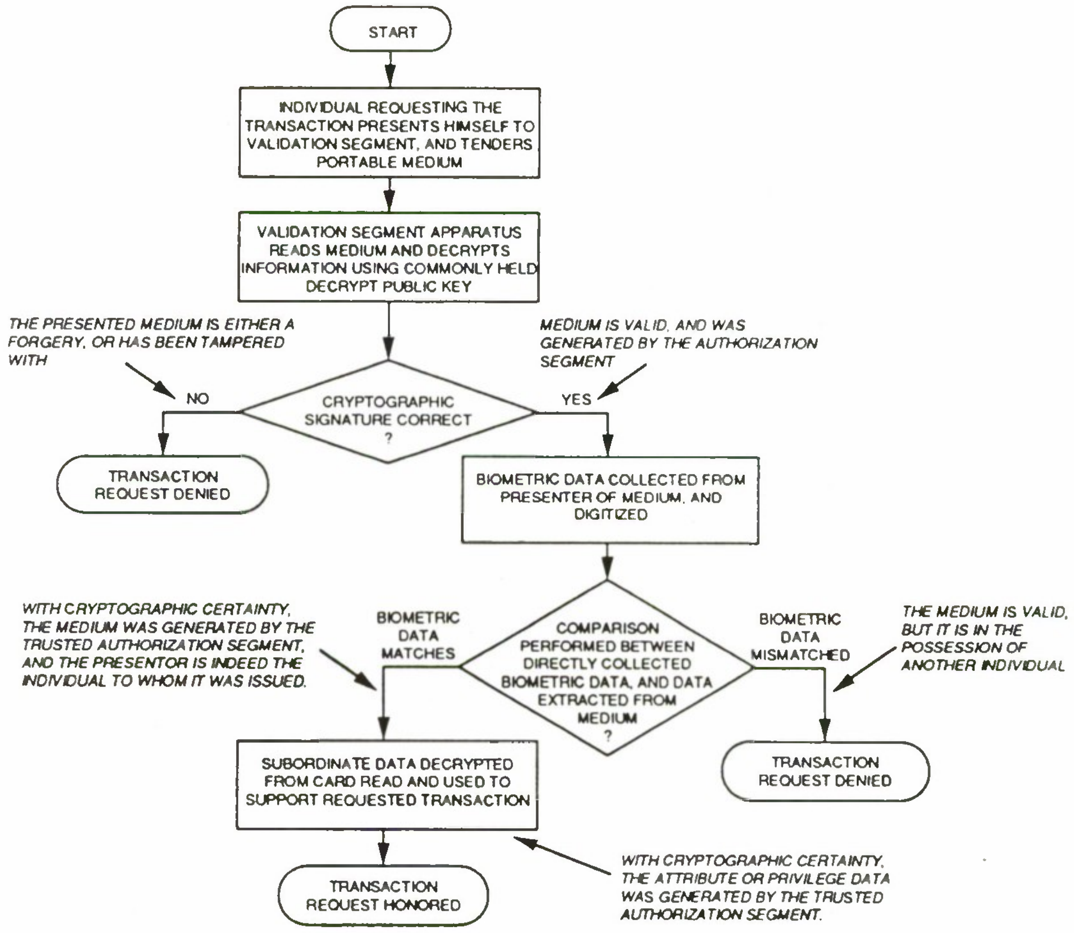

START
INDIVIDUAL REQUESTING THE TRANSACTION PRESENTS HIMSELF TO VALIDATION SEGMENT, AND TENDERS PORTABLE MEDIUM
VALIDATION SEGMENT APPARATUS READS MEDIUM AND DECRYPTS INFORMATION USING COMMONLY HELD DECRYPT PUBLIC KEY
THE PRESENTED MEDIUM IS EITHER A FORGERY, OR HAS BEEN TAMPERED WITH
MEDIUM IS VALID, AND WAS GENERATED BY THE AUTHORIZATION SEGMENT
NO
CRYPTOGRAPHIC SIGNATURE CORRECT ?
YES
TRANSACTION REQUEST DENIED
BIOMETRIC DATA COLLECTED FROM PRESENTER OF MEDIUM, AND DIGITIZED
WITH CRYPTOGRAPHIC CERTAINTY, THE MEDIUM WAS GENERATED BY THE TRUSTED AUTHORIZATION SEGMENT, AND THE PRESENTOR IS INDEED THE INDIVIDUAL TO WHOM IT WAS ISSUED.
BIOMETRIC DATA MATCHES
COMPARISON PERFORMED BETWEEN DIRECTLY COLLECTED BIOMETRIC DATA, AND DATA EXTRACTED FROM MEDIUM ?
BIOMETRIC DATA MISMATCHED
THE MEDIUM IS VALID, BUT IT IS IN THE POSSESSION OF ANOTHER INDIVIDUAL
SUBORDINATE DATA DECRYPTED FROM CARD READ AND USED TO SUPPORT REQUESTED TRANSACTION
TRANSACTION REQUEST DENIED
TRANSACTION REQUEST HONORED
WITH CRYPTOGRAPHIC CERTAINTY, THE ATTRIBUTE OR PRIVILEGE DATA WAS GENERATED BY THE TRUSTED AUTHORIZATION SEGMENT.

Figure 5- Verification Processing

information, emergency medical information, and personal information. Also included on the memory card is data that allows the holder to authenticate himself while he is logging onto a secure computer system. Note that there are multiple records being stored on the credential with each record encrypted using a different key. Also the concept of multiple linked records has been implemented.

Multiple linked records are useful for situations such as a DOD security clearance. The basic clearance information is contained on the first record, with special access or additional clearance information contained in a second record. The general DOD site has the ability to decrypt and use the general information. Only certain selected sites have access to the key needed to decrypt the second tier information. In fact, sites not needing the special access information are unaware that such information is contained on the card.

Finally, the system demonstrates it is possible for a holder of a credential to withhold information from a verification segment if the data recorded on the credential is covered by a personal identification number known only to the valid holder.

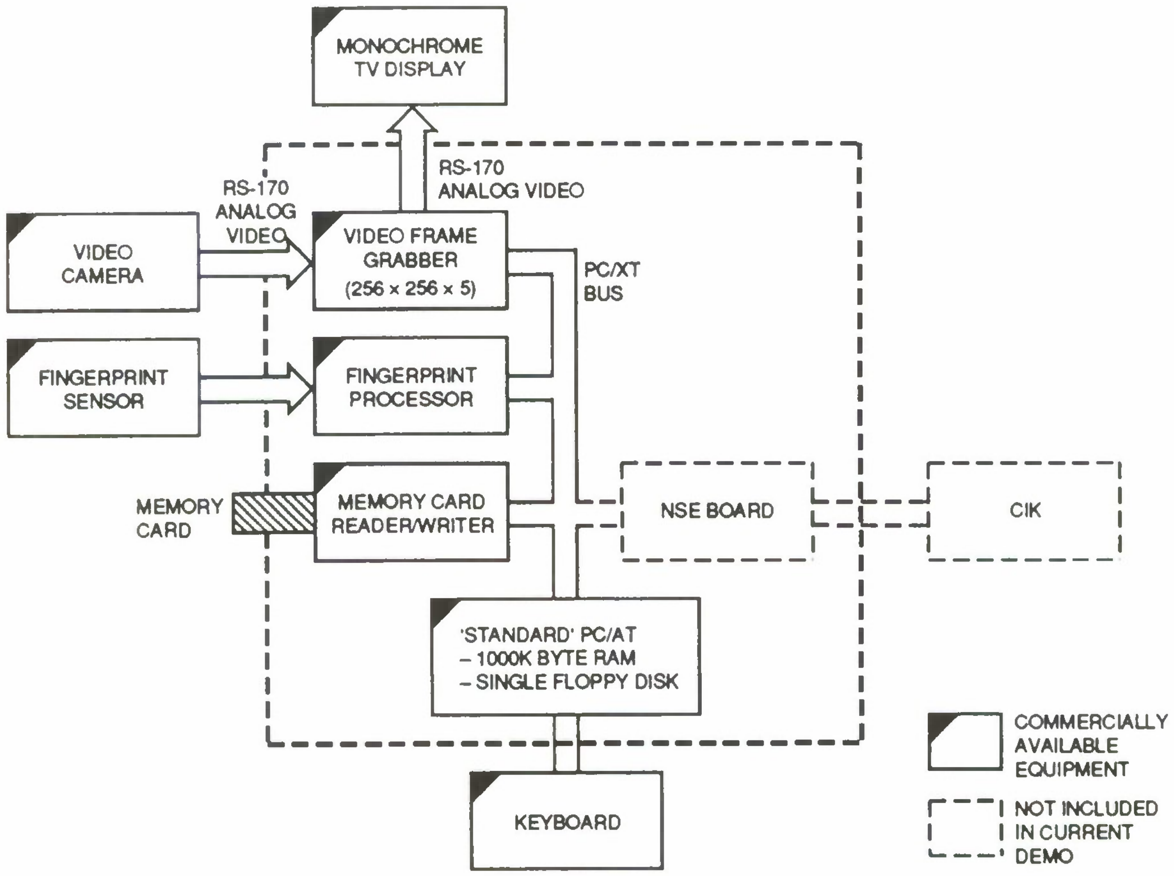

MONOCHROME TV DISPLAY
RS-170 ANALOG VIDEO
RS-170 ANALOG VIDEO
VIDEO CAMERA
VIDEO FRAME GRABBER (256 x 256 x 5)
PC/XT BUS
FINGERPRINT SENSOR
FINGERPRINT PROCESSOR
MEMORY CARD
MEMORY CARD READER/WRITER
NSE BOARD
CIK
'STANDARD' PC/AT – 1000K BYTE RAM – SINGLE FLOPPY DISK
KEYBOARD
COMMERCIALLY AVAILABLE EQUIPMENT
NOT INCLUDED IN CURRENT DEMO

Figure 6- Proof Of Concept System Block Diagram

POSSIBLE APPLICATIONS

The demonstration system is oriented toward an access control or transaction site (credit card) application. The same technique is applicable to other applications, and provides significant benefits when there is a large, mobile group of users, numerous equipments to be used, and difficulty in linking all users and equipments into a centralized and/or on-line data base. Two examples are discussed below, one for STU-III user validation and another for a secure computer network.

STU-III USER IDENTIFICATION

The current STU-III includes an ignition key, assigned to a valid user of the equipment and associated with one particular STU-III. The key, containing a small EEPROM, is carried by the assigned user and inserted into the telephone when a call is placed. The key contains digital data which is read and processed by the STU- III if the key is valid, the secure call is allowed to proceed. As in similar credit card applications, these checks insure that the key is valid, but not guarantee that the holder of the key is the individual to whom it was originally issued.

The proposed extension to this concept involves writing additional biometric data as well as user privilege data onto the ignition key. A compatible biometric sensor would also be added to the STU-III. The voice print is an attractive biometric for this application, since speech digitization and processing is an inherent part of the STU-III architecture. For high security applications, a fingerprint reader is another viable candidate.

Irrespective of the biometric chosen, the valid user's biometric data would be encrypted and stored onto the data key. In addition, a small text file containing his identity, security clearances, etc. would also be encrypted and stored. In operation, the user would insert his key into the modified STU- III, and render his biometric sample. (For voice prints, a phrase would be spoken into the microphone; for fingerprints, the finger would be placed on the sensor plate). The local STU-III Terminal would decrypt the datakey biometric information and compare it to the directly collected information. If matched, the secure call would be allowed to proceed, with the user's text data sent to the destination terminal.

The destination terminal would decrypt the text file at the end of the current call set up protocol, with the resulting information presented via the display. This information would indicate the name and affiliation of the caller, as well as his security clearance levels. The person receiving the call then has cryptographic proof indicating the STU-III from which the call was placed (part of the current STU-III approach), biometric proof that the person placing the call is the assigned holder of the datakey, and cryptographic proof as to the attributes and characteristics of the caller. These benefits are obtained with minimal hardware impact to the current STU-III (possibly adding a biometric sensor and additional ROM space) and appear compatible with the current cryptography and call set- up protocols.

Note that this concept can be extended to allow a mobile user to place a call from any STU-III, rather than in the current scenario where the data key is associated with one specific telephone.

SECURE NETWORK LOG-ON SYSTEM

For this application, it is desired that the requestor validate himself to the computer and that the computer validate himself to the requestor. To accomplish this, valid users would be enrolled into the system and given a portable medium such as a datakey or memory card which would hold their biometric data.

The requestor to machine validation is similar to the STU-III case discussed above. The machine to user validation is accomplished by the machine obtaining a set of encrypted text (in principle, a user unique sort of password) from the credential, decrypting it, and displaying the resulting plain text to the user. If displayed correctly, the user knows that the machine possesses the proper key needed for the validation, and is therefore a valid machine. This decrypted password could also be tied in to an audit or transaction recording system to provide a cryptographically secure proof that the transaction did occur.

In this application, it may be desireable to store a large privilege vector along with the security clearance information. This information is then used by the machine for discretionary access control decisions, to allow access to certain data bases on a selective read or write basis, and other similar uses. Again, the unforgeable and cryptographic basis of this concept permits the user to convey his privileges to a distributed processing network without a central data base or distributed directory. The user carries his directory information around with him.

CONCLUSIONS

The current state of the art in biometrics, public key cryptography, and low cost memory cards allow a revolutionary breakthrough in non-forgeable credentials. The ability to own a credential that is entirely non-forgeable, certifiably correct, and immune to being lost or stolen certainly has some virtue in a society such as ours. The demonstration system proves that the technology to accomplish this is available today, for such low end applications as a department store credit card station, to high security access control points. This type of technology is bound to have an impact to secure communication and secure network technology as well.

REFERENCES

Russell L. Maxwell; Larry J. Write, "A Performance Evaluation of Personnel Identity Verifiers", A report provided by Sandia National Laboratories, July 1987.

AN AUDIT TRAIL REDUCTION PARADIGM BASED ON TRUSTED PROCESSES

Zavdi L. Lichtman and John F. Kimmins

Bell Communications Research
444 Hoes Lane, RRC-1L-217
Piscataway, NJ 08854

1. Introduction

Most audit trail mechanisms record a variety of about 20-40 [1,2,5] types of events, generally providing a pre-selection feature based on event-type and services for post-selection and manipulation of the audit trail data, possibly in real-time. In current systems, pre-selection of events is performed based strictly on event type and/or user-id with no consideration to relations between events.

Automated analysis of an audit trail includes statistical and rule-based methods with respect to a maintained database of user profiles of past activities [4]. The motivation for the reduction paradigm is to reduce the amount of data to be analyzed, without any degradation in the quality of the analysis. If redundant lower level events are removed in a consistent manner, the quality of the anomaly analysis might even improve while reducing the load on the analysis process/machine.

This paper presents a paradigm for audit trail reduction, which is composed of an informal (but sufficiently complete) model of a computing environment, and a list of reduction rules. The reduction rules can be employed either as a pre-selection process or as a post-selection process. Employing the reduction rules as a pre-selection process means that the rules are applied to each event generated, before it is written to the audit trail. Employing the reduction rules as a post-selection process means that the rules are applied to the audit trail after it was generated, producing a reduced audit trail.

This paradigm evolved from a feasibility study for developing an intrusion detection system for a Bellcore application system which primarily performs transaction processing. Analysis of the available audit data revealed that many of the lower level events in the audit trail were redundant.

Section 2 presents a model of a "typical" computing environment, and an audit trail mechanism with the events generated. Section 3 presents a situation analysis based on various events, and the resulting reduction rules. Section 4 shows that the reduction paradigm is also adaptable for more advanced computing environments which include multi-level security (MLS) [3].

2. Computing Environment and Audit Trail Mechanism

This section presents a set of assumptions which together constitute an informal model of a computing environment and its audit trail mechanism. These are required in order to present the audit trail reduction rules. Readers might find some of the assumed audit trail mechanism features (e.g., every event contains the process-id of the parent-process) to be non-existent in current computing environments. Most of the assumed features, however, already exist in experimental or new systems [2,5]. The model presented for the computing environment enables presentation of the reduction rules in the simplest possible way. Most of the assumptions about the computing environment can be modified at the expense of making the reduction rules more complicated.

2.1 Computing Environment

The following is the computing environment:

a. A "typical" multi-user computing environment, used for general computing and/or transaction processing. In particular, no multi-level security (MLS) is assumed; this subject is addressed later.

b. For simplicity of the reduction paradigm, it is assumed that users are allowed a single login and only a single active interactive session. The reduction paradigm requires a data structure of two lists and a boolean switch per interactive user session. The above assumption enables the maintenance of only one such data structure per user. It permits emphasizing the reduction techniques and avoiding the complexity of managing multiple data structures per user.

   If multiple logins and multiple sessions are allowed, logins by the same user must be differentiated with additional attributes such as line number, terminal number, or login time; and sessions within the same login must be differentiated by session number.

c. Users are either privileged (for example, root or superuser in a Unix® system) or not privileged. Privileged users can login either as privileged or non-privileged. Changing privilege requires a login-like process. This ensures that the audit mechanism knows the correct current status of every user.

Network security issues including auditing of network activity are not addressed in this paper.

2.2 Command Classification

User commands and transactions (either line oriented or form oriented) are generally of two types:

a. Commands transferred by the command-line-interpreter directly to the kernel for execution. Such commands do not spawn any process, but might cause terminal events (i.e. events which cannot spawn other events) which are auditable. Examples of such commands include commands to change the working directory, or display the date.

b. Commands which spawn a process. This process can spawn many more subprocesses and terminal events. Examples of such commands include transactions in a transaction system, or a mail command which automatically invokes an editor.

We assume that all user commands (and their parameters) and/or the main processes spawned by user commands generate auditable events which are recorded in the audit trail.

We are interested in characterizing user activities/commands that can spawn in the audit trail many events (sometimes hundreds) that are not required for anomaly analysis. This is typical in transaction processing systems, but also can happen in general computer systems.

2.3 Processes

Initially it is assumed that processes are either trusted or not trusted. Later the concept of relative trust in an MLS environment is introduced.

Unix is a registered trademark of AT&T.

Trusted processes have the following two properties:

a. They reside (as programs) in files that cannot be modified by non-privileged users.
b. They are trusted to obey all the system's security policies, and never violate these policies.

Untrusted processes are processes which evolve from programs that can be modified by non-privileged users. Therefore, they are not trusted to obey all the system's security policies.

It is also assumed that processes cannot be modified in memory while running or waiting for an event, while waiting to gain CPU access, or while being swapped to a disk (either because the swap-area is trusted or it is impossible to "catch" them in the swap-area).

These simple assumptions and definitions can be changed, at the expense of making the "trusted" predicate in the reduction rules more complicated.

The definition for trusted program can also be simplified, at the expense of some risk. For example, transactions in transaction systems might be comprised of application programs and general operating system utility programs. Generally, a deployed transaction machine does not contain the source code of the application programs. In this situation, it is possible to define application programs as trusted, and utility programs as untrusted even if no source code is available for them on the deployed machine. The rationale behind such a policy is that it is possible to replace a utility program with one which contains a Trojan horse, but it is more difficult to do this for an application program.

2.4 Audit Trail Mechanism and Events

The audit trail mechanism generates events. All generated events include (at least) the following information: event-type, object-id(s), user-id, process-id, parent-process-id (when available, otherwise same as process-id), success/failure, date, and time. Process-id is generally a unique identifier or number assigned to each process when it is created. As mentioned in Section 2.1, a single login and a single active session per user are assumed. Otherwise, an additional attribute to differentiate among multiple logins and a session-id are also required.

When a process is initiated, it is certainly possible to find out if the process is trusted or not. For simplicity of the presentation, it is assumed that the process-id and parent-process-id in an event contain an attribute indicating whether or not they represent a trusted process.

Events can be classified along a few dimensions. The following are the event classifications and event-types which are used by the reduction rules:

a. *Login and logout events:*

   These are important because they introduce/delete a user to/from the computing environment, and therefore require special actions by the reduction rules.

b. *Process-events versus non-process events:*

   There are two kinds of process-events: start-process and end-process. Processes are the only subjects (other than users) that can perform actions leading to auditable events, so their creation and deletion is important. Examples of non-process events are object creation, object access, object deletion, etc.

c. *Initiation by a user versus a process:*

As mentioned in Section 2.2, user-commands are considered very important audit data. For completeness, assume that user command events and non-process events spawned by user commands transferred directly to the kernel, have a process-id of the user's command-line-interpreter.

d. *By success/failure:*

Failed events are always recorded. All events have a success/failure indication. We assume that failure of an event is a potential indication of an attempted security violation (even if the event is spawned by a trusted process), or it might be related to a user error which serves as a good indication of user behavior.

3. Audit Data Reduction

3.1 Situation Analysis and Possible Policies

As mentioned, we are interested in characterizing commands that can spawn in the audit trail many events (sometimes hundreds) that are not required for anomaly analysis. The simplest case is when the main process spawned by a user command and all the subprocesses are trusted, and all events are successful. Then, only the original user command (with all its parameters) is needed for anomaly analysis, possibly with the start-process and end-process events of the main process invoked by the command.

In the above case, all events except those at the top level are redundant. In general, the definition of which events are redundant is a matter of policy related to the way that untrusted processes and failed events are viewed. The following two cases specify possible policies describing which events are to be considered non-redundant when an untrusted process is spawned, or an event fails.

a. *An untrusted process is spawned.*

There are a few possible policies concerning which events should be recorded. A reasonable one is the following: Record the start-process and end-process events of the untrusted process and all events spawned by the untrusted process. It is not necessary to record events spawned by trusted subprocesses (these are considered redundant). The rationale is that it is normal for a user command or transaction to spawn both trusted and untrusted processes.This is a chosen policy. It is possible to adopt stricter policies similar to the ones adopted in situation b for a failed event.

b. *An event fails.*

As mentioned, this is a potential indication of an attempted security violation. A few alternative policies are possible. The two policies handled by the reduction rules are:

Alternative 1:

Record the failed event, and then start to record all events for this command/transaction. This means that events following the main start-process event until the failed event are not recorded in the audit trail, except for events of untrusted processes as described in case a.

Alternative 2:

All events relating to the current transaction should be recorded in chronological order. The strategy and required data structures for this alternative are described in the next section.

3.2 Required Data Structures

Alternative 1, for handling a failed event, requires only a simple switch per interactive user session, for indicating when all events are to be recorded. The switch is turned on after a failure, causing all events to be recorded until the command/transaction terminates. At this point, the switch is turned off.

Alternative 2 requires that all the events of a transaction be recorded in chronological order when an event fails. This requirement implies that two lists must be maintained: one for potentially redundant events i.e. events that will be redundant if no event fails in the course of executing the transaction, and a second list for events that are to be recorded due to untrusted processes. The events due to untrusted processes cannot be written directly to the final audit trail, because if a failure occurs the two lists have to be merged in a chronological order.

If an event failure occurs, the two lists are merged, written to the final audit trail, and from that point all events (for this user and the particular session) are written to the final audit trail. If no event fails, then only the second list is written to the final audit trail upon completion of the transaction. Alternatively, in order to eliminate the merge operation, all events can be written to the first list. However, the data structures and the reduction rules are presented for a first list which contains only the potentially redundant events. This first list is merged with the second list when a failure occurs.

The complete data structure has the following three components:

a. SW is a switch to indicate when all events must be recorded, following a failed-event. SW=on means record all events. SW=off means no automatic recording.

b. L1 is a list for recording potentially redundant events. This list is used only for Alternative 2 of failed event processing. It is recorded in the audit trail only when an event fails.

c. L2 is a list for recording events due to untrusted processes. This list is used only for Alternative 2 of failed event processing. If it is not empty, it is always recorded in the audit trail whether an event fails, or at the completion of the transaction.

Both lists are initialized to () (the empty list) every time an event of a user-command is detected.

Note that if simple sequential recording is performed in a multi-user system, then the chronological order of the entire audit trail might be incorrect (although this can be fixed). The chronological order per user, however, is correct.

3.3 Reduction Rules

The reduction rules are given assuming that events are either read from an existing audit trail (a post-selection process), or acted upon when generated by the audit mechanism (a pre-selection process). As mentioned in Section 2.1, for simplicity of the reduction paradigm, it is assumed that users are allowed a single login and only a single active interactive session. The above assumptions enable maintenance of only one data structure per user, emphasizing the reduction techniques and avoiding the complexity of managing multiple data structures per user.

The decision about the action needed for a given event depends on the following:

a. Event-type (process, non-process, user-command, login, logout).
b. The value of the switch.
c. Success or failure of the event.
d. Whether the process-id represents a trusted process.
e. For a process-event, whether the parent-process-id represents a trusted process.

This calls for a multi-dimensional decision table, or a complicated state machine, or a complicated tree-structure or if-statement. After some experiments, a simple rule-list was derived. For every event, rules are tried <u>in order</u> and once a rule succeeds the next event can be processed.

Predicates and selectors are used freely and they are self explanatory. The write(event) operation means writing of the event to the final audit trail. In order to make this process complete, all actions of privileged users are recorded. This is a customary precautionary measure because privileged users can modify trusted programs. The reductions rules are presented in two versions, one for each of the two alternatives for failed event processing.

<u>Rule-List per Event (Failed event processing Alternative 1)</u>

1. If login(event) then create SW for user-id(event); write(event).
2. If logout(event) then delete SW of user-id(event); write(event).
3. If privileged-user(user-id(event)) then write(event).
4. If user-command(event) then SW:=off; write(event).
5. If SW=on then write(event).
6. If failed(event) then write(event); SW:=on;
7. If non-process(event) & trusted(process-id(event)) then do nothing.
8. If non-process(event) & not-trusted(process-id(event)) then write(event).
9. If process(event) & trusted(process-id(event)) & trusted(parent-process-id(event)) then do nothing.
10. If process(event) & (not-trusted(process-id(event)) or not-trusted(parent-process-id(event))) then write(event).

Note that the success of rule 9 for a main process spawned by a user command depends (according to our assumptions) on whether or not the process of the user's command-line-interpreter (which is the parent process of this spawned process) is trusted or not.

Alternative 2 for a failed event requires the use of two lists, as described in Section 3.2. Write(LIST) means writing the entire list to the audit trail, and merge(L1, L2) means merging the two lists based on the time stamps. Note that the assignment of the null list to L2 in rule 6 is done in order to simplify rules 2 and 4. It enables performance of a write(L2) in rules 2 and 4 without a check of the SW. L2 is empty if SW=on, and it is non-empty if SW=off.

<u>Rule-List per Event (Failed event processing Alternative 2)</u>

1. If login(event) then create SW and lists L1 and L2 for user-id(event); write(event).
2. If logout(event) then write(L2); delete SW & L1 & L2 for user-id(event); write(event).
3. If privileged-user(user-id(event)) then write(event).
4. If user-command(event) then write(L2); SW:=off; L1:=(); L2:=();write(event).
5. If SW=on then write(event).
6. If failed(event) then write(merge(L1,L2)); write(event); SW:=on; L2:=().

7. If non-process(event) & trusted(process-id(event)) then add event to L1.
8. If non-process(event) & not-trusted(process-id(event)) then add event to L2.
9. If process(event) & trusted(process-id(event)) & trusted(parent-process-id(event)) then add event to L1.
10. If process(event) & (not-trusted(process-id(event)) or not-trusted(parent-process-id(event))) then add event to L2.

3.4 Examples

In order to demonstrate the working of the reduction rules two examples are given, with and without a failed event. The events are given as a list of events triggered by a specific transaction for a specific user. The following information is given for each event: number (can be viewed also as a time stamp), event name/type, process-id, parent-process-id, Trusted/Untrusted (T/UT), Success/Failure (S/F). The trace shows the number of the rule triggered and the action taken for each alternative. The same rule number is triggered in both alternatives. The actions are for the transaction trans1, the actions for the previous and next transactions are not shown. Finally, the reduced audit trail for the transaction is shown.

Example 1 - Without failed events

#	Event	proc-id	parent-id	T/UT	S/F	Rule#	Alt1-Action	Alt2-Action
1	start-proc trns1	100	shell*	T	S	4	init;write(event)	init;write(event)
2	start-proc	101	100	T	S	9	--	add to L1
3	open file	101	101		S	7	--	add to L1
4	read file	101	101		S	7	--	add to L1
5	start-proc	102	101	UT	S	10	write(event)	add to L2
6	write file	102	102		S	8	write(event)	add to L2
7	start-proc	103	102	T	S	10	write(event)	add to L2
8	write file	103	103		S	7	--	add to L1
9	write file	103	103		S	7	--	add to L1
10	end-proc	103	102	T	S	10	write(event)	add to L2
11	end-proc	102	101	UT	S	10	write(event)	add to L2
12	end-proc	101	100	T	S	9	--	add to L1
13	end-proc	100	shell	T	S	9	--	add to L1
14	start-proc trns2	104	shell	T	S	4	init	write(L2);init

* It is assumed that the shell is a trusted process.
"init" means SW:=off for Alternative 1, and SW:=off; L1:=(); L2:=() for Alternative 2.

The final reduced audit trail is composed of the following events: {1,5,6,7,10,11}. It is the same for both alternatives for handling failed events, since no failure occurred.

To demonstrate the difference between the two alternatives for handling failures, the same sequence of events is used, but event number 9 fails. Obviously, the trace is identical to Example 1 until event number 8.

Example 2 - With failed events (Events 1-8 as in Example 1)

#	Event	proc-id	parent-id	T/UT	S/F	Rule#	Alt1-Action	Alt2-Action
9	write file	103	103		F	6	write(event)	w(m(L1,L2));w(e)*
10	end-proc	103	102	T	F	5	write(event)	write(event)
11	end-proc	102	101	UT	F	5	write(event)	write(event)
12	end-proc	101	100	T	F	5	write(event)	write(event)
13	end-proc	100	shell	T	F	5	write(event)	write(event)
14	start-proc trns2	104	shell	T	S	4	init	write(L2)#;init

* w(m(L1,L2));w(e) = write(merge(L1,L2)); write(event)
# L2 is empty, it was set to () when processing event 9.

Note that it is assumed that the failure of event 9 is propagated back through the end-proc events following it. But it is the first failure that affects the reduction, subsequent event failures have no consequences. The final reduced audit trail for Alternative 1 is composed of the following events: {1,5,6,7,9,10,11,12,13}, and for Alternative 2: {1,2,3,4,5,6,7,9,10,11,12,13}.

4. MLS and Relative Trust

In the previous two sections it was assumed that a process is either trusted or not, and that users are either privileged or not. A user cannot change his privilege without a login-like process. The situation is similar in Multi Level Security (MLS) schemes [3].

In such systems there exists the notion of a Trusted Computing Base (TCB), which contains absolutely trusted programs and data files. Users may be assigned a range of security levels. Each user logs in at one specific security level, and a change of the security level requires a login-like process.

What does all this mean to the predicate "trusted" used in the reduction rules?

Obviously, processes evolving from programs in the TCB are trusted. We also need not be concerned about the relations between processes and other objects. These fall under the basic assumption (stated in Section 2.3) that trusted processes never violate the security policies. There is the question of when is a process (which evolves from a program not in the TCB) to be trusted with respect to a specific user.

For a process evolving from a program which is not in the TCB, the predicate "trusted" succeeds if the process evolves from a program which is relatively trusted with respect to the user. A program is relatively trusted with respect to a user if the user, when attaining his/hers highest security level, cannot modify this program.

Therefore, the reduction paradigm is also suitable for MLS computing environments, with some modifications and adaptations of the assumptions and predicates.

5. Conclusions and Further Research

The reduction paradigm described is certainly not a unique one. It is based on a set of assumptions, and definitions which probably require modifications in order to fit a specific environment. Of course, the critical predicate used in the reduction rules is the "trusted" predicate. The definition and implementation of "trusted" must be carefully evaluated in each computing environment.

The goal of the paper is to convince the reader that current pre-selection features (based exclusively on event type and user-id) are insufficient for audit trail reduction, and might be harmful by removing critical low level events. A reduction paradigm which takes into account relations between events, the amount of trust attributed to processes, and success/failure of events, is needed for a meaningful reduction with no harm to the quality of the anomaly analysis.

A detailed model of the reduction process CPU time requirements and the data recording (to disk) time requirements, and probably some experimentation, are needed to determine the practicality of a reduction paradigm for pre-selection. If the extent of the overhead is too high, the method is adequate only for post-selection.

References

[1] D. E. Denning, D. L. Edwards, R. Jagannathan, T. F. Lunt, and P. G. Neumann, "A Prototype IDES - a Real-Time Intrusion Detection Expert System", Computer Science Laboratory, SRI International, 1987.

[2] C. Dowell, "System V/MLS Security Audit Trail (SAT) and Computer Watch", A presentation at the 4th Intrusion Detection Workshop, Baltimore, MD, October 10, 1989.

[3] M. Gasser, Building a Secure Computer System. New York: Van Nostrand Reinhold, 1988.

[4] T. F. Lunt, "Automated Audit Trail Analysis and Intrusion Detection: A Survey", Proceedings 11th National Computer Security Conference, Baltimore, MD, October, 1988, pp. 65-73.

[5] J. Picciotto, "The design of an Effective Auditing Subsystem", Proceeding 1987 IEEE Symposium on Research in Security and Privacy, Oakland, CA, April 1987, pp. 13-22.

THE COMPUTERWATCH DATA REDUCTION TOOL

Cheri Dowell
Paul Ramstedt

AT&T Bell Laboratories
1 Whippany Road
Whippany, New Jersey 07981

Abstract

This paper presents the design of the first commercial software package that assists the security officer in monitoring a system security audit trail. Developed by the Secure Systems Department at AT&T Bell Laboratories, the ComputerWatch Audit Trail Analysis Tool provides both audit trail data reduction and intrusion-detection capability.

The ComputerWatch Tool reduces the amount of data viewed by the security officer without the loss of any informational content. This enables security officers to focus their attention on areas they are most concerned about as possible avenues of security compromise. The detection mechanism highlights, in report format, the system activity that could indicate possible security-related compromises.

INTRODUCTION

This paper describes the design of AT&T's ComputerWatch Audit Trail Analysis Tool - an add-on package to the secure System V/MLS Operating System.

System V/MLS is a B1-evaluated version of UNIX® System V that provides multi-level security features that comply with the National Computer Security Center (NCSC) orange book B1 security criteria.

One of the security requirements for a B1-evaluated operating system is that it provide an audit trail that records all security-relevant events occurring on the system. The amount of data generated by such an audit trail can get quite large and thus, difficult for a system security officer (SSO) to monitor the activity and interpret it in a timely manner.

The ComputerWatch tool assists the SSO by reducing the amount of data viewed without loss of informational content. It does this by providing a mechanism for examining different views of the audit data based on information relationships. This enables the SSOs to focus their attention on areas they are most concerned about in terms of security-related compromise.

Although the ComputerWatch tool was designed for the System V/MLS audit trail, the tool can easily be modified to operate on an audit trail from another system.

The tool was written to assist an SSO but not to replace him/her. It is instead an expert system approach to summarizing security sensitive events and applying detection rules to generate warning messages highlighting anomalous behavior. It also provides a method for detailed analysis of user actions to track suspicious behavior.

CURRENT SYSTEM

System V/MLS Audit Trail Structure

The level to which events are audited affects both the processing speed and the detection accuracy of any audit trail analysis tool.

The detection accuracy of an analysis tool is limited by the types of data being audited. The System V/MLS Security Audit Trail (SAT) generates an audit record for all security-relevant events and all data accesses. Twenty-five selectable trace channels record the types of security-relevant information shown **in Figure 1.**

Channel	Event	Channel	Event
00	clock sync record	12	IPC access failure
01	fork executed	13	removal of IPC object
02	exec executed	14	user level trace record
03	exit executed	16	file declassification
04	system call failure	17	IPC object declassification
05	file unlink/remove	18	mount/unmount of file system
06	file creation	19	signals sent by root
07	additional link to file	21	creation of unnamed pipe
08	successful file access	22	modification of effective uid or gid of a process
09	file access failure	23	change of owner, group, or mode bits of a file
10	IPC object creation	24	change of owner, group, or mode bits of an IPC object
11	successful IPC access		

Figure 1. System V/MLS Audit Channels

Since the audit trail for a B1-rated system can cause significant impact on performance, a major design goal of a good security audit trail should be to minimize performance overhead by using a compact record format.

In System V/MLS, the overhead of the audit trail is *less* than 4%. This is achieved by double buffering in kernel memory to optimize disk I/O as well as using a binary format to reduce individual records to an average size of 16 bytes.

The size of an audit trail varies depending on the types and amount of events being audited. The amount of events being recorded is dependent on the type of machine the data is generated from, the length of time covered in the trail, and the amount of activity occurring on the system. With System V/MLS, it is also a function of the amount of activity being recorded (i.e., the types and number of audit channels turned on).

The storage format of the System V/MLS audit trail is constructed to save disk space. The audit trail structure consists of a header followed by audit records. This header is used as an internal name map for each object in the system (i.e., user, group, label, tty, file system). The audit records represent *deltas* or changes to the original information in the header. The objects in the audit records are represented by their abbreviated names; the actual names are reconstructed during processing of the audit trail by the formatter module (e.g., inodes are mapped to their actual file pathnames).

Although the compact binary record format saves system disk space and decreases the amount of time required to write out the binary records, there is a drawback to the compact form of audit trail. The trade-off is that it takes time to convert the binary data to a human readable format. Because this conversion is usually a one time occurrence, the advantages of a compact format outweigh the drawbacks.

The size of the audit trail buffer also affects system performance and audit trail integrity. By making the audit trail buffer small, the time for writes to disk and the amount of data potentially left in buffers as a result of a system crash is reduced, but unfortunately system performance is severely degraded (i.e., more writes are required to save the same amount of data to disk)).

Finally, because of the sensitive nature of the data in an audit trail, it must be protected from compromise. System V/MLS maintains the integrity of its audit records by only generating records through two secure paths - via the secure system kernel, and through a trusted user-level interface.

Security Feature

Super-user access is required for the `ComputerWatch` program to access the System V/MLS audit trail data and protect its own results. System V/MLS restricts the super-user to operating at the system level, and to logging in as a regular user for added security protection. In addition, both the SSO, and the terminal that the SSO is using, must be cleared to operate at System High (SysHi), the highest security level on the system.

Operational Scenarios

The following describes how the `ComputerWatch` tool would typically be used:

— The SSO formats and loads the set of audit trail data he or she is interested in analyzing.

— The SSO generates a system activity summary report to get an idea of the types and amount of activity occurring on the system. He/she runs it with detection mode *off* to perform his/her own analysis of what is happening on the system. He/she then, runs the report with detection mode *on* to see how the tool evaluates the system activity.

— Based on the results highlighted in the summary report, the SSO runs several standard queries against the audit data to isolate the activity of individual users on the system. The SSO then determines which user(s) are responsible for the security-relevant activity that looks suspicious.

— If the SSO detects some disturbing events as a result of running queries triggered by the results of the summary report, he or she may decide to execute several queries against the individual user. If the results of the queries targeted to a single UID show abnormal behavior, the SSO may decide to reload several files of data from a previous day's SAT files.

— As a result of evidence collected by the SSO concerning a particular UID, the SSO may decide to create several custom queries to keep a close track on future behavior exhibited by this particular user.

In addition, the SSO can shape the tool to fit his/her environment and needs by performing the following tasks:

— The SSO can modify the format of the System Activity Summary Report to suit local needs.

— The SSO can modify or add to the detection rules used to highlight values and produce analysis messages in the summary report; He/she can tune the rules to best detect suspicious activity on his/her particular system.

— After perusing the summary report and the results of several provided queries, the SSO may decide to build a custom query to view the audit data. Using this important feature, the SSO creates an extension to the basic set of queries to satisfy special needs.

The `ComputerWatch` tool includes sample cron scripts that allow the user to execute the tool in a batch mode out-of-hours to ease the performance impact. Cron scripts are routines that allow the user to program the machine to run a job at a particular date and time or on a regular basis. The cron scripts can get data from another machine, format and load the data, and send to a printer, a summary report analyzing the events occurring in the audit trail. This enables the SSO to pick up the summary report, scan it (perhaps, first thing in the morning), and decide if further audit data study is needed.

User Interface

The user interface is an important part of any auditing tool. If it is awkward or difficult to learn, it will quickly be abandoned in day-to-day operations.

The user interface for the `ComputerWatch` tool was constructed using the AT&T ETIP Designer™ Package (ETIP stands for Extended Terminal Interface Prototype) to create a hierarchal structure of menus. This off-the-shelf utility features pop-up menus, built in choice selection, and both function key and arrow key movement to provide an intuitive feel. The ETIP Designer also provides a character-based interface allowing the package to run on a variety of different terminals.

The ETIP Designer places menus on the screen to conserve space, but allows the user the capability to change the size, shape, and screen location of the menus. The user can leave menus on the screen and traverse between them or bring up a new menu at each invocation.

At installation time, the user specifies default parameters that can be overridden at execution time. On terminals that have programmable function keys, the program downloads a pre-defined set of functions to the user's terminal keyboard making maneuvering through the menus easier.

Dataflow Diagram

Figure 2 shows a dataflow diagram of the components of the `ComputerWatch` tool. Each of the components will be discussed in following sections.

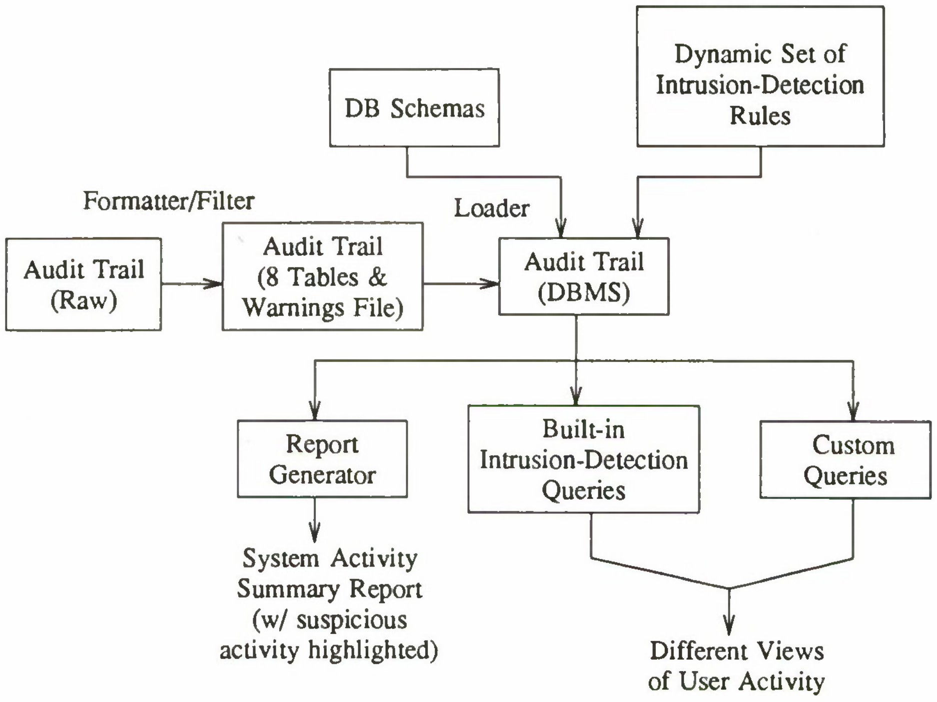

DB Schemas
Dynamic Set of Intrusion-Detection Rules
Formatter/Filter
Loader
Audit Trail (Raw)
Audit Trail (8 Tables & Warnings File)
Audit Trail (DBMS)
Report Generator
Built-in Intrusion-Detection Queries
Custom Queries
System Activity Summary Report (w/ suspicious activity highlighted)
Different Views of User Activity

Figure 2. Dataflow Diagram of `ComputerWatch` Components

Formatter/Filter Module

System V/MLS generates an audit trail made up of raw binary data. The `ComputerWatch` formatter converts the raw data into eight human-readable database files. As part of this conversion process, key fields are indexed for faster data retrieval. Analysis of audit data determined that eight tables were optimal for queries based on the various combinations of data items most frequently referenced together. The following lists the eight files and their contents:

— the *exec.tab* file contains process execution information.

— the *fork.tab* file contains process fork/exit information.

— the *alias.tab* file contains listing of all files that were accessed and have links or alias names.

— the *ipc.tab* file contains interprocess communication information (i.e., message, semaphore, and shared memory read/writes).

— the *syscall.tab* file contains system call failure information.

— the *uli.tab* file contains user level record information (i.e., logins, password changes, printer disabling, changes in user clearances and privileges).

— the *io.tab* file contains all read/write success and failure information.

— the *other.tab* file contains the remaining audit trail information not fitting the data characteristics of the previously mentioned files (i.e., mounts, umounts, kills, chmods, chgrps, chowns, setuids, setgids, links, unlinks, un-named pipes, mknode/creates, and reclassifications).

In addition to the eight audit files, a WARNINGS file is produced containing any unrecognized record formats found in the audit trail. Strange records in the WARNINGS file may indicate that someone is tampering with the integrity of the audit records.

The most time-consuming part of the analysis tool is the conversion of binary audit data to DBMS format. In terms of speed, on a AT&T 3B4000, it currently takes on the order of four minutes to format one megabyte of binary audit data. Fortunately formatting is a one-time event, and formatting can be done off-line using a cron script.

Although the `ComputerWatch` tool was designed for the System V/MLS audit trail, it can easily be modified to operate on an audit trail from another system. `ComputerWatch` was written as separate, independent modules. By modifying a single module, the format/filter module, the tool can be made to handle a new format of audit trail.

Loader Module

A loader is provided to select and load the particular set of previously-formatted audit data to be analyzed.

The SSO selects which system to view, and uses the loader to link the necessary set of audit data into a work directory along with the schemas which interpret and provide structure to the data.

DBMS Module

The `ComputerWatch` utility comes with a small relational data base management system (DBMS) that runs under the UNIX Operating System. Emphasis in building the DBMS was placed on ease of use, and making it simple to understand and maintain. The query language under the DBMS is SQL-like. The types of query operations provided include:

— join - joining of multiple tables along a common field or set of fields.

— project - selecting particular fields or columns of a table.

— select - selecting particular rows of data from a table.

— index - indexing on particular fields for faster data retrieval time.

— asort - sorting in descending/ascending order by field.

— dist - calculating totals, averages, and maximum and minimum values of fields.

— print - printing out resultant tables.

While there is no true project command, the select command performs both selection and projection. A simple query may be answered by executing one of the DBMS commands. However, there is frequently a need for queries that require a sequence of these commands. The shell language provides the means to build complicated transactions from simple DBMS commands. These complicated transactions can be built by:

— having commands execute singly in sequence with output stored in an intermediate file to serve as input to the next command.

— using a shell procedure consisting of a sequence of DBMS commands which will execute as if a single command had been given.

The DBMS code size was kept to a minimum to be able to place a level of trust in the code. The ComputerWatch DBMS only includes the database operations that should be used on an audit trail. It does NOT contain data field modification routines because they are not necessary and could be used to compromise the audit trail data.

The DBMS operates on flat data files that get their structure from schemas. The advantage of flat files is their interpretation can easily be changed by modifying the DBMS schemas. Also, this allows the DBMS to operate on files from other machines or ones generated by a UNIX System editor.

System Activity Summary Report

The purpose of the System Activity Summary Report is to provide a summary of the security-relevant activity happening on the system (i.e., activity that causes a user to gain or modify his/her access privileges or activity that causes the privileges associated with an object to change). It can indicate what types of system events need a closer look on the SSO's host machine(s).

The System Activity Summary Report operates in two modes - detection mode *on* or *off*. Running with detection mode *on* causes a set of intrusion-detection rules to highlight areas of concern in the report and to send explanatory messages to an analysis file. Running with detection mode *off* allows the SSO to perform his/her own analysis on the audit data.

Figure 3 is an example of a summary report (Note: The report format can be modified to meet site-dependent needs).

```
                          ACTIVITY SUMMARY REPORT

 DATE: Thu - August 31, 1989     TIME: 09:24 AM    SYSTEM: Mars

Logins:                       Successful        Failed               % Failed
                                  5               7                     58

                                                   5 Known User(s)
                                                   2 Unknown User(s)
--------------------------------------------------------------------------
Processes:                    # spawned         # exited
                               258               248
--------------------------------------------------------------------------
File Accesses:                Successful        Failed               % Failed
                              1203               27                     2

                               971 Read(s)        2 Read(s)
                               232 Write(s)      25 Write(s)
--------------------------------------------------------------------------
TCB Accesses:                 Successful        Failed               % Failed
                              1165               27                     2
--------------------------------------------------------------------------
Superuser Activity:           SU's Failed       SU's Successful   Setuid Execs
                                  1                5                220

                                                   2 Root           181 uid=Root
                                                   3 Non-Root        39 uid<>Root
--------------------------------------------------------------------------
User Reclass. Activity:       # of attempts     # at system
                                  2                1

                                  1 Failed
                                  1 Successful
--------------------------------------------------------------------------
File Reclass. Activity:       # of attempts     # at system
                                  0                0

                                  0 Failed
                                  0 Successful
--------------------------------------------------------------------------
New Objects:                  # created         # at system
                                 37                34
--------------------------------------------------------------------------
Chmod:                        # setuids         # at system
                                  0                0
--------------------------------------------------------------------------
Lps:                          # outputs         # classified
                                  0                0
--------------------------------------------------------------------------
Mounts:                       # of mounts
                                  0
--------------------------------------------------------------------------
```

Figure 3. Sample `ComputerWatch` Summary Report

The following shows the detection messages that would be output if the report program was executed with detection mode *on*:

(58%) Too HIGH - Percentage of failed logins
(28%) Too HIGH - Percentage of unknown users
        attempting access
(10) Too HIGH - Number of Non-exiting processes
(50%) Too HIGH - Percentage of failed newprivs

**** False HIGH - Number of failed newprivs*
(0) Too LOW - Number of successful chprivs

The SSO can add, delete, or modify any of the textual field descriptions in the report. The user can also delete or move around the data item fields. Adding new data field items is a planned future enhancement.

There are three basic levels of detection statistics (system, group, user). Statistical information for each event system-wide is provided by the previously discussed summary report. Statistical information for each event based on users is provided by the detection queries which are discussed in the next section. Statistical information for each event based on user groups will be a future enhancement.

There is some controversy over whether viewing statistics at a system level can detect intrusions. For some systems, the values may be too erratic to derive much from them in terms of detection. We have found it to be useful in showing what areas do not require attention rather than what areas do. For example, since little or no file declassification is evident, declassification obviously does not need more careful study. The ComputerWatch tool can maintain a different copy of the summary report for each machine being analyzed and it has been found that in some cases, the typical activity of a machine forms a recognizable pattern.

Queries Module

The detection queries provided are designed to assist an SSO in detecting "simple" system security breaches involving intrusion, disclosure, and integrity subversion. The queries were designed to display similar security-relevant system activity as that shown in the summary report, but at a user-level.

There are two types of detection queries provided by the tool:

— Queries that output the uid of users and the number of times they caused the occurrence of a security-relevant event (i.e., uid event_count);

— Queries that output detailed information about a particular user and security-relevant event (i.e., Process ID, terminal, date, time, User ID, Group ID, event, event-objects);

The detection queries provided by the product package are as follows:

1. **Failed LOGINS** - For all users or each login ID.
2. **Failed SUS** - For all users or each login ID.
3. **Failed NEWPRIVS** - For all users or each login ID.
4. **Failed CHPRIVS** - For all users or each login ID.
5. **Failed FILE ACCESSES** - For all users or each login ID.
6. **Successful LOGINS** - For all users or each login ID.
7. **Successful SUS** - For all users or each login ID.
8. **Successful NEWPRIVS** - For all users or each login ID.
9. **Successful CHPRIVS** - For all users or each login ID.
10. **EUID=ROOT** - For all users or each login ID.
11. **USER Session Query** - Display entire user session.
12. **WHO modified a GIVEN FILE** - For any individual file.
13. **FILE ALIASES** - For any individual file.

The ComputerWatch Tool provides the user with the capability to design his/her own queries for intrusion-detection. An SQL-based query language is provided for this purpose.

User-defined queries can be targeted to the standard tables as well as to temporary tables created by custom queries. The following sample query gives information about system users who have executed any given system command. Note that the command and a threshold value are passed as shell variables to the query making it very flexible:

```
select uid from exec.tab into s1.tmp where file = "$1"
dist count by uid in s1.tmp into s2.tmp
select uid from s2.tmp into s3.tmp where count gt "$2"
asort -r count in s3.tmp into syscom.tmp
print syscom.q
```

For example, by filling in a menu form it is possible to run this query with the arguments `query /bin/who 3` to see which users have executed the "who" command more than 3 times. Similarly, filling in `query /bin/ps 0` displays users that have executed the "ps" command. Both types of queries are important in detecting intruders; intruders often will check to see if anyone else is on the machine they have gained access to and logoff if someone else is logged on. Intruders also frequently check to see what their activity looks like to other users on the system by running the "ps" command. It is also used to check that they have left no processes running that could indicate that they ever occupied the system.

Rules Module

The SSO has the ability to do his/her own analysis of the System Activity Summary Report or to have the `ComputerWatch` tool provide him with an analysis. A set of user-modifiable and user-tunable detection rules are provided with the tool that highlight areas of the System Activity Summary Report that can be of concern from a security perspective.

Rules fire (or execute) when a given equation is satisfied and the rules in their predecessor list have fired. The firing of detection rules causes a value in the Summary Report to be highlighted in a particular color and/or generates an analysis message.

The following lists the fields that make up a rule:

1. **rule id** - a unique number used to identify a rule.
2. **active?** - an on- or off-bit indicating whether the rule is capable of being fired. The user can use it to temporarily turn off a rule.
3. **rule type** - indicates the type of equation that should be satisfied in order for the rule to fire.
4. **screen box id** - indicates a box to highlight in the summary report if the rule fires.
5. **threshold** - a threshold value used in the equation to be satisfied by the rule.
6. **predecessor list** - a list of rules that must fire before the current rule fires.
7. **message** - message to be output if the rule fires.
8. **equation values** - indicates the statistical value fields in the summary report to be used in the equation.

There are 5 rule types which operate on the specified value(s). The equations associated with the rule types are as follows:

1. value > threshold.
2. value < threshold.
3. ((value1 / (value1 + value2)) * 100) >= threshold
4. value1 - value2 >= threshold
5. Always true.

The rules are contained in a separate data file and executed such that:

— Rules that depend on other rules must have their predecessors fire and have their equation be satisfied before being evaluated.

— Error-checking prevents the creation of rules with a predecessor list that would result in a loop.

A detection rules editor is provided to enable a user to create/modify/delete/list rules, and to better tune the rules provided by modifying thresholds to fit the characteristics of a particular host machine. A different set of rules is maintained for each machine to be analyzed.

FUTURE SYSTEM

There are three levels of statistical observation that will ultimately be provided by the `ComputerWatch` tool (system, user, group). Each statistical level will have its own set of detection rules and profile characteristics.

Ongoing development of the tool will include the analysis of network activity as well as that of a single system. The System V/MLS Trusted Network Utility (TNU) already outputs audit trail records that are capable of being analyzed by the `ComputerWatch` tool.

The next major release of the tool will feature both batch and real-time execution modes. A security workstation will be able to monitor and apply intrusion detection rules to audit trail data as it is being generated by several host systems. The security workstation can either be a separate machine connected to the hosts or be a virtual system residing on one of the hosts.

CONCLUSION

The design of a security audit trail needs to be carefully considered because it can consume large amounts of storage and exhaust much of the power of the CPU.

Because audit trail data is repetitious, without a means of reducing and analyzing it, a security officer has little chance of finding security compromises. The `ComputerWatch` Audit Trail Analysis Tool can detect anomalies and alert a security officer in a timely fashion.

REFERENCES

[1] *ComputerWatch User's Guide*, AT&T Bell Laboratories, version 1.0.

[2] *Security Audit Trail (SAT) Design Document*, AT&T Bell Laboratories, version 1.2.1.

[3] *System V/MLS Trusted Facility Manual (TFM)*, AT&T Bell Laboratories, version 1.2.1.

[4] Final Evaluation Report: *System V/MLS Release 1.1.2*, CSC-EPL-89/003, October 18,1989.

ACKNOWLEDGEMENTS

The `ComputerWatch` product came about due to hard work and conversations with Howard Israel, Bill Leighton, Jon Weiss, and others too numerous to mention. Funding for the project was provided solely by AT&T.

Analysis of Audit and Protocol Data using Methods from Artificial Intelligence

Dr. Winfried R. E. Weiss
Adalbert Baur

Siemens AG
ZFE IS SOF4
Otto-Hahn-Ring 6
D-8000 Munich 83,
West-Germany

Abstract:

Protocol data are generated in many application areas by computer systems. In most cases, it is impossible to analyze the resulting huge amount of data without computer support. In this report, we discuss the principles of an AI-based tool for the anlysis of protocol data, which we have implemented. Although being general in nature, the tool will first be used for analyzing audit data generated by secure computer systems.

The tool was designed with flexibility and ease-of-use in mind. Flexibility is provided by allowing users to define the incoming data format as well as the evaluation criteria. Users may link actions to evaluation criteria which will be executed if the criteria is satisfied. All user definable items are entered via a menu based human interface.

1 Introduction

There are many (computer) systems that generate some kind of protocol data. The mechanism producing the data is usually called audit or protocol mechanism, the data are called audit or protocol data. In most cases the amount of data produced is so large that it is impossible to analyze the data by hand.

Computer systems satisfying the criteria C2 or higher of the Orange Book [DoD 1985] must have an audit mechanism which records every security relevant action. Similar requirements are defined in the IT-Sicherheitskriterien [ZSI 1989], the German equivalent of the Orange Book. These audit mechanisms are usually distributed with some kind of analysis tool, since the Orange Book and the NCSC guide to auditing [NCSC 1987] requires this. However, the functionality of these tools is mostly very restricted. They support only the data analysis on a record-by-record basis.

Related Work
More advanced tools are described by T. Lunt and D. Denning [Denning 1987], [IDES 1988] and by Liepins [Liepins 1989]. There statistical analysis is used to detect anomalous user behaviour, working on the premise that anybody abusing a system will show abnormal user behaviour. An survey of existing analysis tools is given in [Lunt 1988].

Pupose of the Analysis Tool
We describe a Protocol Data Analysis Tool (PDAT) that uses methods from artificial intelligence to analyze protocol data very thoroughly. The analysis tool is designed such that it can be applied for almost every system generating protocol data. The are only few requirements that the audit data have to fulfil.

Since secure computer systems from different manufacturers generate audit data with very different formats, a major aspect while designing the PDAT was its configurability. Thus a very flexible tool was designed. PDAT is in fact so flexible and powerful that it can be used for analyzing not only audit data but almost any kind of protocol data. Protocol data are generated during the auditing of secure computer systems, test analysis, diagnosis, optimization, validation and operational control.

Implementation details have been left out of this report in favour of discussing requirements and showing how they are fulfilled by the PDAT.

Terminolgy
Let us now clarify some terminology. There will be some kind of setup that is monitored. This setup is called "system", the monitoring mechnism is called "audit mechanism", the information generated is called "audit data". Something or somebody acting in the system will be called either "user" or "process".

The program described in this report doing the analysis will be called PDAT. The person analyzing the protocol data using the analysis tool will be called the "operator" (of the analysis tool).

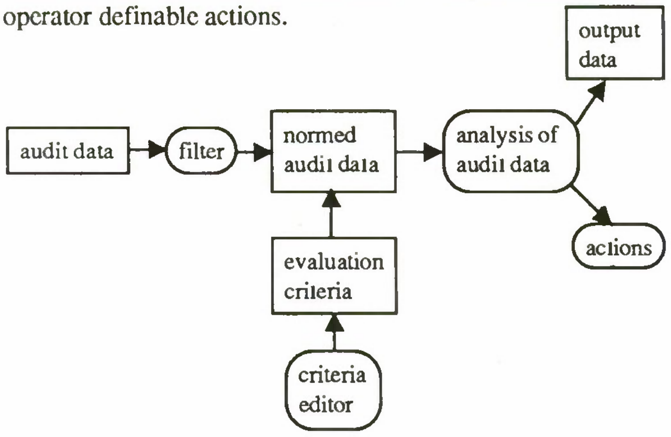
The terminology in this report is taken from the analysis of audit data generated by secure computer systems. The reader should always keep in mind that this is only an example and that the analysis tool is applicable in much more general cases.

Overview
In section 2 we describe the architecture of the PDAT. Section 3 and 4 describe the configurability and human interface. Sec-

tion 5 describes the different types of evaluation criteria. Reactions the PDAT can take and the different work modi are described in section 6. Sections 7 and 8 provide a summary and present an outlook into possibilities which will be explored in the future.

2 Architecture

The following picture shows the data flow in the PDAT. Audit data are transformed into the internal format. They are then analyzed by applying criteria which have been defined and stored in the data base. Satisfaction of any criterion leeds to operator definable actions.



Format of the Audit Records
There are only very few assumptions made about the format of the audit data. It is assumed that the data come as a sequence of records each one describing a relevant event for the system.

The audit records can be described best by saying that they must have a structure similar to variable records used in Pascal. It is not assumed that all records have the same format. Records can look different depending on the information stored in the record itself.

The records have to be in the same logical order in which they are to be analyzed. Usually this means ordering according to the time when the event described by the record took place. But any other form of ordering is definable by the operator.

By saying that each record describes an event we mean that each record contains the logical information about one event. Events are the smallest logical entities that are recognized by the audit mechanism.

For the example of analyzing audit data this means that each record contains all logical information about a single security relevant event executed by the system. Thus the name of the action, user, time, object, success or failure have to be recorded along with any other important information. A record in the internal data format might look like:

[(user,bob),(action,login),(time,7:34),(terminal,p7), (success,failed)...]

This record describes an unsuccessful login attempt of the user bob at 7:34 on terminal p7.

In a secure environment it is the operators responsibility to ensure that the data generated by the audit mechanism are transferred securely to the analysis tool. We do not provide for this because almost every secure system does have mechanisms to ensure this.

3 Configuration

A major aspect of the PDAT is its configurability. This was already mentioned when we described the record format. But obviously such a tool has to provide flexibility in other aspects as well.

The PDAT is able to analyze audit data from different systems which are set up in different environments. Thus the PDAT has to be able to be configured. Special demands stemming from the different environments have to be met. The better one can adjust the PDAT to ones special circumstances, the more useful will the analysis tool be in supporting the analysis of the audit data.

When using the PDAT to analyze audit data of secure computer systems, the security administrator is thus able to reduce the number of false alarms (i. e. reducing the amount of incidents the security administrator has to check) to a minimum while still having a high detection rate of real system abuse.

There is an other aspect to configurability. Obviously applying the PDAT will use resources. Manpower will be needed to check the deteced abuse of the system. Computing resources - i. e. CPU time, memory, disk space etc. - will be needed to run the PDAT. When considering secure computer systems, this means loss of preformance of the system when the PDAT is running.

Thus configurability empowers the operator to adjust the thoroughness of the analysis. The more thorough the analysis, the more resources are needed. The system administrator can do a risk assessment before configuring the tool. When configuring the tool he can measure the resources needed and can make a cost versus effectiveness analysis. He can thus tune the performance of the tool to the requirements.

In an open research environment one may just want to detect outside break-in attempts. This will be possible with the use of few resources.

In a highly secure system, on the other hand, containing sensitive data one may be more interested in the internal abuse of the system. Banking systems for example suffer most losses by legitimate employees abusing their rights. It may even be possible that in such system external break-in attempts are ruled out by organisatorial procedures or other control mechanisms like smart cards etc. . Thus making it unnecessary to look for outside break-in attempts. But one may want a very thourough analysis of the internal threats. The security administrator may in this case decide to spend a considerable amount of the resources on the analysis to detect abuse and prevent losses of capital or to ensure the integrity of the information on the system.

Configurability is supported by the menu-guided interface of the PDAT which is described in the following section.

4 Human Interface

All logical constructions described in section 5 are operator-definable. This can be best explained by considering the following example.

Assume you want to search for the occurence of certain events. This means that you have to search the data for records that confirm to a certain specification. Then the analysis tool does two things:

First it provides a menu-guided interface for describing requirements for the records that are to be selected. These requirements are thus completely operator-definable. Each description of requirements is stored under an operator-given name by the analysis tool. There are directories and paths under which named criteria can be stored like in many ordinary hierachical file systems.

The second step is applying criteria which have been defined beforehand. Here the operator has to tell the analysis tool which list of previously defined criteria is to be checked for occurrences against the actual audit data.

Defining new criteria in a system for a special application will be a major part of the work. It is planned to include sample configurations for some of the more common applications to facilitate the configuring.

Obviously configuration is crucial for the success of the analysis tool. But it is not sufficient to offer the possibility of configuring the system. One has to make configuring as obvious and intelligible as possible. Thus the human interface is of major importance. Special care has been taken to make these menus self-explicable and easy to understand. However, for each menu there is a help facility describing the workings of the particular menu.

5 Types of Evaluation Criteria

In this section we will describe the different types of evaluation criteria offered by PDAT. We will start by describing the simplest evaluation mechanisms which can be characterized as recordwise selection.

5.1 Selecting Records Satisfying Certain Criteria

The basic selection mechanism is record based, but goes far beyond the capabilities of a UNIX grep over a file pipe.

First the user interface will be much more comfortable. The tool allows the logical description of fields in the record. The contents of the fields can then be described by metacharacters. To select all unsuccessful login attempts one would specify a selection criterion as follows:

Select all records where the action field contains login and the success field contains failed.

There will be a menu where the operator just has to fill in the contents of the fields.

From now on such a description will be called a "selection criterion".

Next, one will be able to combine selection criteria using normal logical operators. Thus new selection criteria can be built up in easy steps from simple selection criteria, making it easier to generate exactly the right selection criteria.

Defining the right selection criterion is very important since selection criteria form the basis of all of the following analysis methods.

5.2 Dynamic Table and Static Data Base

The analysis tool relies on a static data base and a dynamic table to store information about the system. The data base contains static information about the system which is rarely changed. It can be used to store the home directory, full user name, address, telephone number, normal working hours, times of absence for every user (e. g. holidays, or business trip), public holidays etc. . This data base can only be changed by the operator of the tool himself.

<u>Dynamic Table</u>

The dynamic table, as indicated by the name, is dynamically updated by the tool depending on the contents of the analyzed records. Any tool for analyzing data must be able to store information about the state of the system that is to be analyzed. This is mandatory since it is impossible to store all information about the state of a system at a certain point of time in every single record.

Taking our example of analyzing audit data, it is likely that only relative paths are given for all objects referenced in a record. Thus the analysis tool has to maintain the current path for any active process. This means maintaining some kind of internal table which contains every active process and its corresponding current path.

We will give a simple example for an application using the dynamic table. Assume the operator wants to know whenever somebody accesses a file /bin/admin/secure/secret. The selection criterion would then have to be defined using the dynamic table as follows:

Look up the current path of the process accessing a file, compute the absolut name of the file using this current path and the relative path found in the record. The selection criterion becoming satisfied when the resulting name is /bin/admin/secure/secret.

A simple dynamic table storing only the relative paths of all processes might not be enough. Other things like currently opened files or other objects, access rights, etc. may have to be stored.

The information needed to be kept in the dynamic table depends on the audited system. It has thus to be configured by the operator. He can define which information is kept in the dynamic table by the PDAT.

Note that it is not suffcient to define what is to be kept in the dynamic table. One also has to define rules how to update the dynamic table. This means looking for records which contain information that will change the contents of the dynamic table.

For our simple example, in which only the current path is maintained, this would mean selecting all records that contain the action change_directory. The information of each of these records then has to be used to update the dynamic table.

Definition and update rules are stored by the PDAT. The operator can define different dynamic tables and can specify which one to use in the actual analysis.

5.3 Searching for Behavioural Patterns

In many cases it is not sufficient to be able to select single records. One may wish to look through the data for (a single occurence) of a pattern consisting of several records. i. e. one is looking for a sequence of records describing the pattern.

In a bank system for example one may look for the following pattern:

Step 1: Somebody transfers a large amount of money internally to his account.

Step 2: A few days later the money is transferred back.

In this situation there will be no money missing in the bank accounts, but somebody has illegally collected quite a lot of interest.

A more complicated example in a system containing secure information is the following:

Step 1: One user opens a file

Step 2: A second user opens the same file

Step 3: The first user writes into the same file

Step 4: The second user reads from the file.

This sequence of actions may indicate an attempted illegal communication.

The situation is made more difficult by the requirement that arbitrary many records can lie between the different steps of the behavioural pattern. The importance of this requirement can be seen when considering our example of the analysis of audit data of a multi-user environment.

A point worth keeping in mind is that several instances of the same behaviour criterion can be active at the same time. Consider the case where one record for user1 fits the first step in a behaviour and the next record, a record for user2, fits the first step as well. Then both records could be the beginning of a behavioural pattern in which the operator is interested.

Solution

After having discussed the problem we want to describe the solution, which will take into consideration all of the points above. Behaviour criteria are described as a succession of arbitrarily many steps. Each step is defined as a selection criterion in the simplest case but may again be a behaviour. Once the first step of the behaviour is satisfied, all relevant information about this is stored in the internal data base. Again it is up to the operator to specify which information is relevant.

Whenever there is an entry in the data base saying that the first step of the behavioural pattern has been found the analysis tool is looking for records satisfying step 2.

It is crucial that the analysis tool still searches for records satisfying step1, because there might be another sequence of records satisfying the behaviour criterion, running concurrently with the first one. Thus each record is checked against all behaviours that are currently in the list of criteria applied by the operator.

When describing behaviours that run concurrently it is necessary to include variables in the definition of behaviour criteria. Thus it must be possible to define a variable called U1, that is to contain the value of the field containing the user name in the record satisfying the first step of the behaviour.

The effect of this can be demonstrated in our example. Imagine we find a record satifying step1 for user1. We then have to define a variable containing the name of the file. Then we look for a record satisfying step2, where we have to check that the filename is identical to the contents of the variable from the first step, but where the user name is different.

Defining the behaviour criterion one can thus say: Look for a record satisfying step2 where the contents of the field object is identical with the contents of the variable object1 defined in the first step, and where the contents of the field user is not identical to the variable user1. This allows us to select records describing a behaviour, where all records describe actions on the same file.

Complex Case
Of course this is only a special and simple case. More complex cases can be defined by using logical operators on the variables. Variables can thus be set depending on the information in the records. Records can then be selected depending on the contents of the variables defined beforehand. It is not only possible to test for identity, but it is also possible to test the contents using metacharacters.

Again care has been taken that arbitrarily many of these behaviours can be checked at the same time. These can be many copies of the same behaviour or copies of different behaviours containing the same selection criteria in different steps. The internal storage in the data base enables unique identification of behaviours and variables. Each behaviour criterion for which the first record is found, is automatically given a new set of variables.

The behaviour criteria are created and then stored under an operator defined name in the data base, as we already described for the case of selection criteria. Again a menu-guided interface greatly eases the task of defining behaviour criteria for the operator.

In general it is possible to define behaviour criteria whose steps can either be selection criteria or behaviour criteria that have been defined beforehand. Moreover, any step can be defined by a logical conjunction or disjunction of such previously defined criteria.

Obviously it is also possible to select information about the dynamic table and static data base for selecting records which satisfy the requirements for one step.

Stop Criteria
There is a problem of size here. The number of started criteria can grow quickly. Therefore it is required to define so called "Stop Criteria" for each behaviour. Essentially these criteria are used for describing in which cases a started behaviour can no longer become fulfilled.

In the above case of the bank, a stop criteria could be a quaterly revision of all accounts. If nothing suspicious has been found in this revision, all behaviours that started before this revision may be stopped.

In the second case of illegal communication, a stop criterion would be if user1 does a logout before user2 accesses the file in question.

5.4 Statistical Analysis

Statistical methods play a major part in analyzing audit data. The analysis tool can be used for statistical analysis which is based on a sequential work-through of the audit records. The statistical method used are operator definable.

It is possible to analyze the number of occurences of selection or behaviour criteria as well as analyzing the contents of records satisfying certain criteria.

To see the usefullness of statistical analysis consider the following example. Measure the percentage of unsuccessful login attempts among all login attempts. This is relevant when searching for break-in attempts. Usually the number of unsuccessful logins will be below 10%, mostly resulting from people mistyping their password. If the percentage suddenly increases to over 99% a break-in attempt is almost certain.

In a second example one might be interested in the login times of the users. One would have to define a selection criterion selecting all successful login attempts. The statistical method would be to find the average login time and the variance of it. A great discrepancy between the actual login time and computed average login time may indicate that an illegal user has logged in under a legal user name.

Statistical methods can also be used to discover the use of covert channels by for example detecting high rates of file creation and deletion.

5.5 Learning Normal System Behaviour

Based on the statistical analysis of audit data, "normal behaviour" can be derived from the audit data. This can be defined on a per user basis as well as for the entire system.

Thus one can derive the average working hours of a user. The PDAT can then detect for example that a certain user almost always works between 7.00 a.m. and 5.00 p.m. . A login for this user at 5.00 a. m. would be regard as not normal by the PDAT.

To use this method the operator has to define a selection, behaviour or statistical criterion which is to be used to determine the normal user or system behaviour. Thus the PDAT can record the average number of processes during the different times of the day. This can then be averaged over a longer period of time. The average would be weighted so that the more recent operating days would be of greater importance.

Unusual activity like many more active processes at a certain time might be an indication that something unusual, like a worm invasion, is going on. By using weighted averages over the past, the system will continually update its knowledge about normal behaviour. For any such criterion one has to define the difference to the normal behaviour that is considered acceptable. Anything above the acceptable limit will be reported.

Simple statistical analysis as described in the previous section is not sufficient to perform this task. Additional methods like trend analysis and methods from artificial intelligence are used to be aware of changes in the behaviour which are interesting for the operator.

5.6 Time Considerations

Obviously time is precious when analyzing large amounts of audit data. The operator therfore is given the choice to define a set of evaluation criteria. When applying such a set of criteria obviously all criteria are checked at the same time.

An advantage of these sets is, that one can "compile" them to get an optimal search order for the criteria. Finding the right search order is a problem wich grows exponentially with the number of criteria in the set. Therefore it is impossible to figure out the exact optimal solution. One can however use techniques from artificial intelligence to fond an almost optimal solution, which fulfills the speed requirements.

6 Actions and Work Modi

If the PDAT finds any criterion fulfilled it executes an operation called action. This action can be set differently for every applied criterion. There are predefined operations like "write on the console", "write into a file" etc.. An action can also be to start any program outside the PDAT, shuting down terminals etc..

Actions are thus operator-defineable and stored using names just like the evaluation criteria. When applying an evaluation criterion the operator has to define which action is to be taken when this criterion is found to be true.

In principle the action can consist of executing any utility or defined subroutine that has been bound to the analysis tool or calling any program outside the analysis tool.

The tool can be used as an offline analysis tool as well as an online analysis tool. Online meaning, that the audit data are written into a buffer from which the PDAT reads. Offline means the audit data have already been written into a file and are now read from this file.

It is also possible to analyze several different incoming data streams at the same time. This is needed when the operator is responsible for several machines in a network. He is then able to anlyze the audit data from these different machines at the same time.

7 State of the implementaion

A prototyp with reduced functionality has been implemented using a Prolog system. This prototyp is used to demonstrate that the ideas above are realisable.

The menu interface has been implemented on an X-Window based workstation. It is beeing tested by a separate group to assure consistancy and ease-of-use.

At the moment this prototype is tested with data from a secure UNIX operating system.

8 Summary

This report describes an audit anlysis tool that is being developed by the Central Research Laboratories of the Siemens AG., West-Germany. A prototype implementation has been finished. The prototype is used to demonstrate the capabilities and functionality and for performance measurements.

This report is part of the work of the ESPRIT-project Commandos.

Literature

[Denning 1987] An Intrusion-Detection Model, IEEE Transactions of Software Engineering,Denning, D. E., vol SE-13 no.2, 222-232

[DoD 1985] DoD-Studie 5200.28-STD "Department of Defense Trusted Computer System Evaluation Criteria", December 1985, Library No. S225,711

[IDES 1988] A Prototype IDES: A realtime Intrusion Detection Expert System, Teresa F. Lunt and R. Jagamathan, SRI Project ECU 7508, April 1988

[Liepins 1989] Anomaly Detection: Purpose and Framework, Liepins, G. E. and Vaccaro, H. S.,Proceedings of the 12th National Computer Security Conference,495-504

[Lunt 1988] Automated Audit Trail Analysis and Intrusion Detection: A Survey,
Lunt, T. F.,Proceedings of the 11th National Computer Security Conference, 65-73

[NCSC 1987] "Guidlines for Auditlog Mechanisms in Secure Computer Systems" The Aerospace Corporation, May 1987

[ZSI 1989] IT-Sicherheitskriterien, ZSI-Zentralstelle für Sicherheit in der Infomationstechnik, 1. Fassung, 11. Januar 1989

A UNIX PROTOTYPE FOR INTRUSION AND ANOMALY DETECTION IN SECURE NETWORKS*

J.R. Winkler
Planning Research Corporation, R&D, MS: 5S3
1500 Planning Research Drive
McLean, VA 22102 USA
(703) 556-1108

ABSTRACT

Secure systems and networks generate vast amounts of audit information that may reveal unusual situations or patterns of use. While the required analysis is usually performed only after other evidence is uncovered, a strong need exists for real-time analysis. The need is driven by the reality of the situation: the "trusted" user is often the weak link in otherwise trusted systems and networks. Such a situation is referred to as the "insider threat problem." This paper describes a prototype real-time network and host security monitor that supports automated as well as interactive audit trail analysis. Audit records, representing tokens of actual user (or host) behavior, are examined in context of user profiles, which represent expected behavior. The essential problem in the analysis of audit records is the timely correlation and fusion of disjoint details into an assessment of the current security status of users and hosts on a network. In our system, audit records, or indications of actual events, are correlated with known indicators organized in hierarchies of concern, or security status. As indications are matched with indicators, a more detailed examination at the next level of indicator granularity is triggered. Thus, as recognized indicators and/or sets of indicators are matched, concern levels increase and the system analyzes increasingly detailed classes of audit events for the user or host in question. Analysis capabilities include statistical as well as expert systems components. These cooperate in automated examination of the various "concern levels" of data analysis. Cooperation and cross-tasking of statistical and rule-based components is believed to be unique in such systems. The system combines a sophisticated, graphical user interface with a series of analytical tools to provide unprecedented support for monitoring and auditing user and host activity in secure networks.

1.0 Introduction

This paper describes the Information Security Officer's Assistant (ISOA), a functioning UNIX-based prototype for centralized real-time network security monitoring [1,2]. Section 1 is a brief overview of the field of intrusion/anomaly detection and discusses some of the functional requirements for such systems. A high-level technical description of the architecture of the ISOA implementation is presented in Section 2. Section 3 concludes with a description of planned extensions to the ISOA.

The field of intrusion/anomaly detection in secure systems and networks is relatively young, with few related projects reported to date [3,4,5,6,7,8,9]. In systems that process sensitive information, the technical means for implementing security include access controls, sensitivity labeling, and related measures. Once a user is granted access, such "secure" systems only enforce the security policies that they implement. Clearly, technical measures for affecting system protection are not effective where the trusted user is the weak link in otherwise trusted networks [10]. Individuals with normal privileges can do considerable damage as well as misuse their legitimate privileges.

Audit records are often used as a means for warning and for maintaining a record of security relevant events. Typically, such audit information is examined by a systems administrator some time after the events have transpired. Unless the audit record indicates an immediately recognizable security violation, most security relevant situations are difficult to discern. This is due to the overall volume of audit information that is generated. As audit collection granularity increases, the analysis problem becomes correspondingly difficult. At this point the collection, storage, and analysis of audit data incurs the application of significant resources. Problems associated with low level collection and analysis of audit events include:

- Audit data volume — at the finest levels of granularity, audit data for a single user can exceed 10MB of data per day. Common methods for reducing the required storage are compression and selective collection.
- Timely analysis — most audit trails receive at best a cursory examination, often only long after the events have transpired.
- Identifying "Suspicious" behavior — the difficulty in formulating useful definitions of "suspicious behavior" is especially apparent when one examines events that are within the domain of permitted user actions, but suspect when placed in context of the normal behavior for users in the same role.

In order to facilitate the identification of suspicious or unusual behavior, audit events should include more than the date and time of user sessions, or the occasional message regarding failed access to data. Although examining date and time of login can often identify masqueraders [3,7], numerous other measures can identify unusual usage of resources by legitimate users. The identification of abnormal usage and the correlation of diverse events buried in the audit trail presents a nearly impossible situation without the use of automated analysis.

In monitoring events that do not constitute direct violations, it is necessary to have a means for assessing observed behavior. One way that this can be achieved is to specify expected behavior on a per user and host basis. Expected behavior can be represented via profiles that specify thresholds and associated reliability factors for discrete events. Actual observed events then can be compared to expected measures, and deviations can be identified via statistical checks of expected versus actual behavior [11]. However, statistical measures are incapable of identifying situations that can not be identified by monitoring thresholds. In addition, combining individual statistical measures seldom results in a readily comprehended meta-view of the overall security status. It therefore becomes necessary to effect second-order analysis oriented toward correlating and resolving the meaning of diverse events. The application of expert systems technology lends itself to this, since a rule-base can specify the possible relations and implied meaning of diverse events.

In Artificial Intelligence (AI) terminology, a rule-base consists of numerous individual reasoning rules that are encoded in an if-then or condition-action form. Such rules then serve as the criteria for forming conclusions as indicators. The rule-based approach lends itself to the posing of sophisticated queries based on known scenarios or recognized patterns of behavior. Rule-based analysis can be effectively used in both evaluating the meaning of a group of events, and in prospecting for unusual behavior. Where statistical measures can quantify behavior, rule-based analysis can answer conditional questions based on sets of events. By combining statistical and rule-based analysis, the results of statistical measures of activity can be examined to achieve a more encompassing view.

The monitoring and analysis of user behavior in system usage is fundamentally different and outside the domain of technical security measures (access controls, security labels, etc.). Such analysis and real-time monitoring can serve as a powerful adjunct to security mechanisms.

1.1 Functional Requirements

The effective monitoring and analysis of behavior requires both a method of data collection and a strategy for data analysis. The system design, or technical approach to addressing these issues, is dependent on the functional requirements for the gathering and analysis of the audit data. The number of audit records processed, or examined, varies with the level of current activity, with the current collection granularity, and with the current security concern level. At the finest level of granularity, the volume of records becomes overwhelming. It is thus necessary to employ selective collection in order to limit the collection of audit events to a reasonable, manageable level. However, selective collection must be managed and controlled to allow the collection of information at the finest level of granularity, when such information is necessary for critical analysis. Selective collection would be specified best on a per user and host basis.

At increasing levels of granularity, additional kinds of audit events must be captured and sent to the monitoring system. These kinds of events can be organized in various classes with sub-types identified within the classes. One method of controlling the level of collection granularity can be affected by specifying that collection should include, or exclude, an indicated class or type of audit event.

Once audit records have entered the monitoring system, it is necessary to have a strategy for deriving meaning from the vast number of related and unrelated events that arrive over time. Such a strategy for analysis should be flexible, such that the analysis is responsive to the current view of the overall security situation. This entails maintaining an abstract view of the current security relevant actions for each monitored user and host. In view of the volume of data managed and the resulting analytical limits, the strategy should incorporate a means for directing analysis to different levels, depending on the current concern levels and volume of data received. Analysis should be performed in a variety of dimensions. At the lowest level, it is necessary to examine the incidence of outright violations. At higher levels, one can perform various statistical analysis and various rule-based analyses.

Further, the processing involved in statistical and rule-based analysis could be optimized if they are applied in concert. This is in line with the desire to have a capability for the resolution of individual statistical measures. Concerted statistical and rule-based analysis could be realized under the direction of an intelligent process that would need to have an understanding of the meaning of distinct audit events as well as of their possible relations. Optimization would most likely be based upon a framework for analysis that depends on both an organization inherent in the definitions of the audit event classes and sub-types, and a hierarchy of security concern levels. Various schemes for defining hierarchies of security concern levels are possible.

2.0 System Design

The ISOA system design is based on the previous discussion of the functional requirements, and was implemented on a UNIX-based workstation. Numerous processes interact in a complex manner built on interprocess communication (IPC) and sockets. A high-level description of the underlying processing model and some of the system features follows.

As implemented in the ISOA audit records represent tokens of actual behavior that are analyzed and compared with expected behavior as represented by user and host profiles. Each monitored host produces audit records of security relevant events. These audit records are sent to the ISOA for central collection and analysis (Figure 1). The current security status of the network is displayed in a graphical user interface that affords the Information Security Officer (ISO) the capabilities for further interactive analysis as well as for direct control over any host and user session.

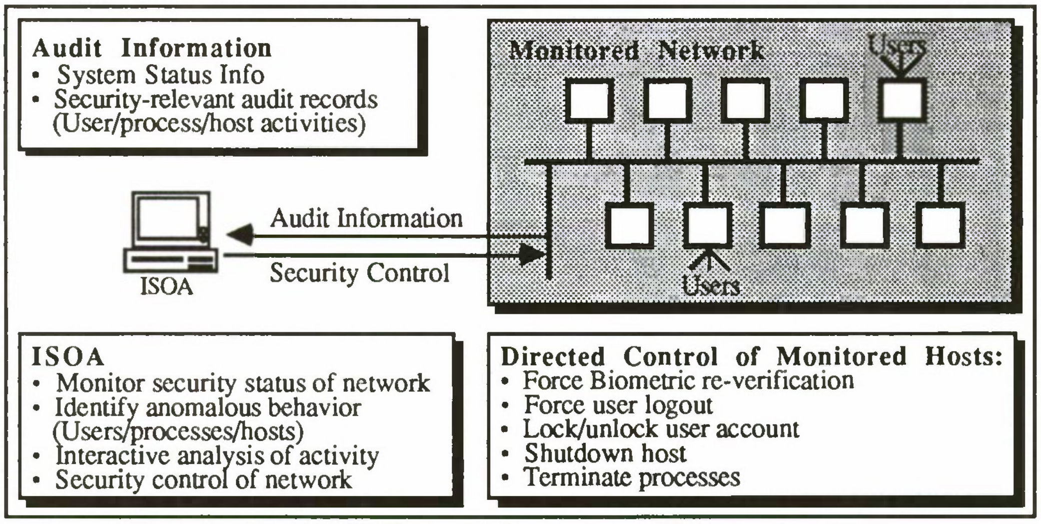

Audit Information
• System Status Info
• Security-relevant audit records (User/process/host activities)
Monitored Network
Users
Audit Information
Security Control
ISOA
Users
ISOA
• Monitor security status of network
• Identify anomalous behavior (Users/processes/hosts)
• Interactive analysis of activity
• Security control of network
Directed Control of Monitored Hosts:
• Force Biometric re-verification
• Force user logout
• Lock/unlock user account
• Shutdown host
• Terminate processes

Figure 1. Central Monitoring and Control

2.1 Overview of Processing Model

In this system ,we have adopted the general Indications and Warning (I&W) model to track events at the level of the individual user and host (Figure 2). The term indicator is used to refer to abstract events that are identified in advance of monitoring. In contrast, indications represent actual occurrences of the corresponding events. In our model, we have grouped both individual indicators as well as sets of related indicators at the user and host levels. These are organized such that as events occur, corresponding indications are triggered or set to the appropriate level of concern.

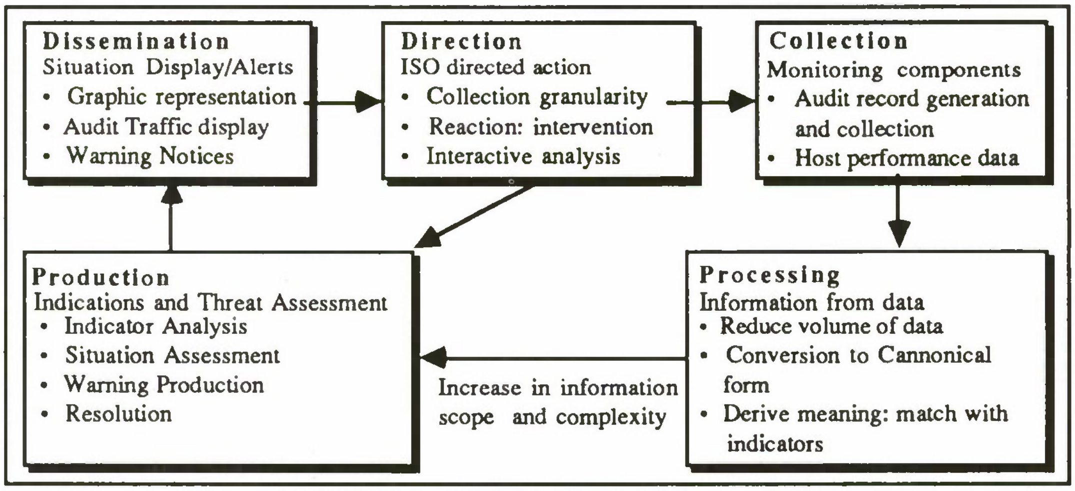


Figure 2. I&W Based Monitoring and Anomaly Detection

One of the difficulties we encountered is that there is seldom a direct match of indicators with real-world events. Perhaps the most obvious class of examples is the set of thresholds that the system maintains. The system receives audit records for a particular event (for instance the UNIX "access" system call with mode set to "read") and maintains a count for the number of these events during a given session. At various points in time, session statistics are calculated against these events in light of the expected measures as specified in the appropriate profile. Since we maintain both user and host profiles, it is possible to exceed a threshold for a given measure for a user, a host, or both. The fact that a given threshold has been exceeded does not in and of itself necessarily indicate that a user is engaged in "suspicious" behavior. Consequently, it is important to organize these indicators to allow the modeling and identification of various classes of suspicious behavior. To this end, we support a number of distinct threat profiles for suspicious behavior (aggregator, imposter, misfeasor, etc.) and a separate means for identifying that overall measures of various events are at unusual levels.

Beyond tracking user and hosts individually, two major classes of measures are defined — real-time and session. Real-time measures require immediate analysis and examination, while session measures require at minimum start-of-session and end-of-session analysis. In practice, session-level measures are examined more often, as driven by the need for resolution.

In summary, the underlying processing model of the ISOA consists of a hierarchy of concern levels constructed from indicators. Analysis is structured around these indicators to build a global view of the security status for each monitored user and host.

2.2 Centralized Monitoring and Analysis

The functioning of the prototype can be seen as the interaction of the audit process (AUDIT), the profile checker (PROCHK), the statistical components (STATS), the expert system (HADES), and various other system components. Briefly, ISOA receives audit records from monitored nodes. The AUDIT process then converts these to a compact, canonical form we call a 'thread'. The term 'thread' is used since related audit records can be viewed as a 'thread of behavior'.

Audit events are organized according to classes of events with each class having a defined set of types of events. For each class and type, a set of valid statuses and associated completion codes exist. Classes of audit events include:

- Log events, includes: login, logout
- System calls
- Data access— a subset of system calls, includes: read, write, append, delete
- Privileged operations
- Unusually privileged operations— operations requiring exceptional privileges
- Node control events, includes: node up, node down, reset clock, lock user account

Each audit event class/type is identified in a common ISOA header file. Each event listed in this file defines: a text description of the event (used by the AUDIT process for generating human readable audit records), a distinct code identifying the event, and a code that controls processing by AUDIT.

After converting audit records to canonical form, AUDIT appends the resulting record to the appropriate thread, performs a table lookup of the audit event, extracts the appropriate text description of the event, formats a human-readable audit record, displays this text audit record, and proceeds to perform additional checks on the event. These additional checks are dependent on the processing code associated with the audit event in the audit table.

As audit records are appended to a given thread, they are reviewed for outright violations (real-time measures or events), which are reported directly to the ISO and broadcast to other analytical components via a standard mechanism. When the processing code indicates that profile checking is required, AUDIT either performs simple profile checking directly, or if complex checking is required, notifies PROCHK. Subsequently, AUDIT and/or PROCHK inform the appropriate system components when significant events are identified. In an attempt to identify suspicious behavior, further examination of the audit records is performed by PROCHK and the analytical components.

The broadcast mechanism consists of an extensible request queue mechanism, and is implemented in shared memory. A standard interface allows any ISOA processes to request specific functions from other ISOA processes. This forms the basis for controlling the current depth of analysis for individual users and hosts. As indications warrant, AUDIT, PROCHK, and/or HADES can request resolution of indications and inter-session profile checking.

In order to effect concerted problem solving, current processing information is maintained globally, listing concern levels for various indicators. Maintaining a per user and per host view of the current security status allows us to define concern levels for individual users and nodes, and identify how individual ISOA processes view these. Naturally, these concern levels vary from process to process. Consequently, an overall resolution strategy is necessary.

2.3 Preliminary Anomaly Detection

Preliminary anomaly detection takes place in real-time during the collection of audit data. Pre-determined events such as login and logout trigger the AUDIT process to notify PROCHK when audit records relating to these events arrive. AUDIT places "request" packets on PROCHK's pending request queue that contain information required to investigate the current indicator or event of interest.

PROCHK will, depending on the type of event, loop though a table of profile data to determine if an analysis is warranted. Analysis is specified by table parameters that can be modified by the ISO via the PROEDT profile editor. If analysis is specified, further table elements are tested against current parameters to check for real-time violations, or to trigger indicators representing deviations from expected behavior. A failed login attempt would constitute a real-time violation, while a login attempt at a time outside the parameters of the user's profile for login times is an example of the need to trigger an indicator. In contrast, a data read threshold exception is an example of the kind of indicator that requires an increase in the current depth of analysis and/or drives the need for resolution.

2.4 Secondary Anomaly Detection

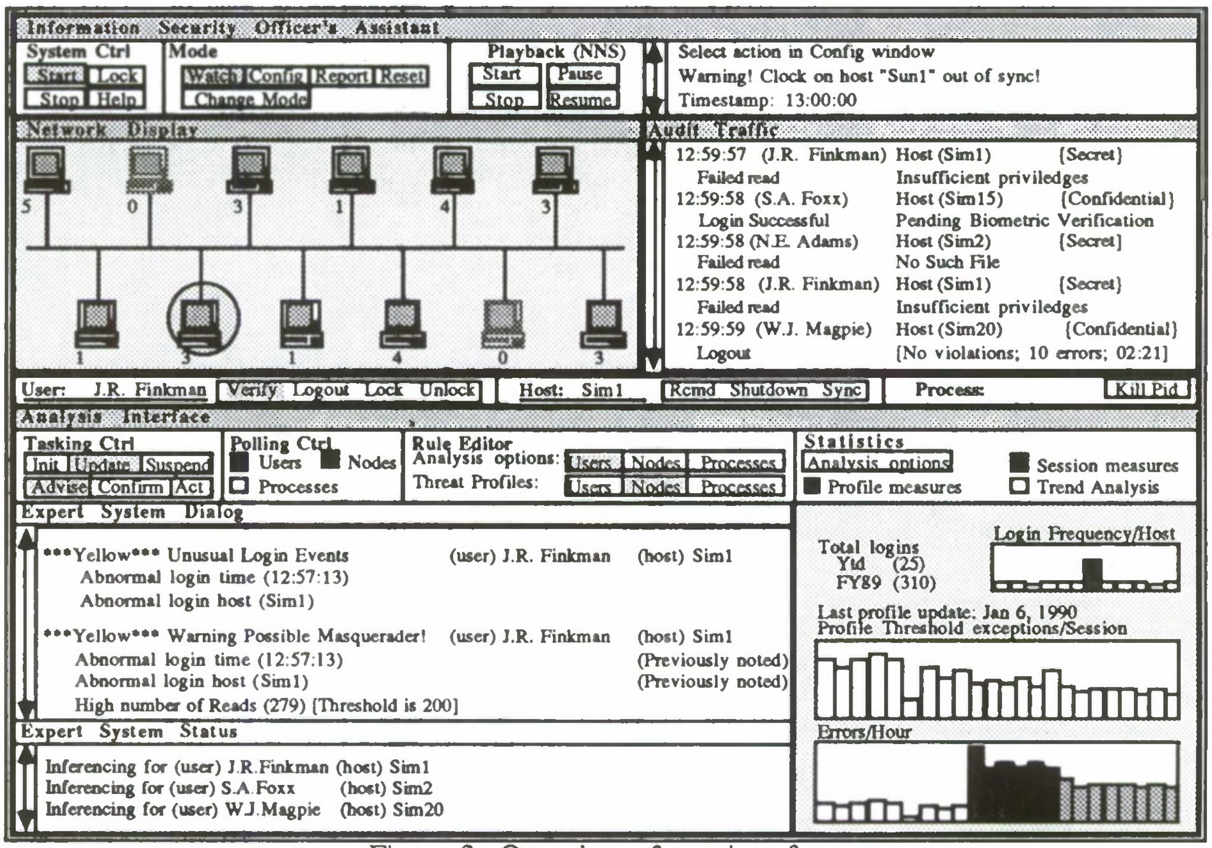
Secondary anomaly detection is invoked at the end of a user login session or when required for resolution. At session end, the current session statistics are checked against the appropriate profiles by PROCHK. Session exceptions are determined in much the same way as PROCHK identifies primary indicators, the difference being the statistics being compared.

Also, while resolving primary indicator states (discussed below), HADES may need more information than is currently available in the form of indicator states. HADES can request PROCHK to perform various sub-sets of session level checking. PROCHK will subsequently signal HADES in the event that such checking resulted in changed indicators. In addition, the ISO can force PROCHK to poll session metrics periodically on a clock basis or at will.

2.5 Anomaly Resolution and Control

As stated above, HADES is notified by AUDIT, PROCHK, and other system components when the state of indicators changes significantly. In essence, HADES attempts to resolve the meaning of the current state of indicators. This is done by evaluating the appropriate subset of the overall rulebase. The rulebase consists of a number of individual rules that relate various indicator states with each other and with established threat profiles. Currently, forward inferencing is used in the evaluation of current security status. If ambiguous situations are encountered, HADES can initiate further low-level indicator analysis by signaling other systems components (most notably, PROCHK and STATS).

The end result of anomaly resolution is presented to the ISO in the form of a graphical alert, advice, and explanation as to why HADES thinks the current security level is appropriate. The graphical interface (figure 3) consists of numerous other windows for monitoring audit traffic, directing control of the ISOA system, and for effecting direct control of monitored user sessions and hosts. As monitoring indicates anomalous activity on a given host, the ISO can obtain more in depth information by using the mouse to click on a graphical representation of the host. Graphical representations of monitored hosts are color coded to depict their current security status.



```
Select action in Config window
Warning! Clock on host "Sun1" out of sync!
Timestamp: 13:00:00
```

```
12:59:57  (J.R. Finkman)  Host (Sim1)            {Secret}
  Failed read             Insufficient priviledges
12:59:58  (S.A. Foxx)     Host (Sim15)           {Confidential}
  Login Successful        Pending Biometric Verification
12:59:58 (N.E. Adams)     Host (Sim2)            {Secret}
  Failed read             No Such File
12:59:58  (J.R. Finkman)  Host (Sim1)            {Secret}
  Failed read             Insufficient priviledges
12:59:59  (W.J. Magpie)   Host (Sim20)           {Confidential}
  Logout                  [No violations; 10 errors; 02:21]
```

User: J.R. Finkman Host: Sim1 Process:

```
***Yellow*** Unusual Login Events          (user) J.R. Finkman   (host) Sim1
   Abnormal login time (12:57:13)
   Abnormal login host (Sim1)

***Yellow*** Warning Possible Masquerader!  (user) J.R. Finkman   (host) Sim1
   Abnormal login time (12:57:13)                                (Previously noted)
   Abnormal login host (Sim1)                                    (Previously noted)
   High number of Reads (279) [Threshold is 200]
```

```
Inferencing for (user) J.R.Finkman (host) Sim1
Inferencing for (user) S.A.Foxx     (host) Sim2
Inferencing for (user) W.J.Magpie   (host) Sim20
```

Total logins
Ytd (25)
FY89 (310)

Last profile update: Jan 6, 1990

Figure 3. Overview of user interface

A significant characteristic of this system is the monitoring and control of remote hosts on a network. Locking user accounts, killing processes, forced logouts, re-synchronizing monitored system's clocks, and forcing shutdown of remote monitored hosts from ISOA, are a few of the functions performed.

4.0 Future Directions and Summary

By comparing the statistical measure of a user's past behavior with their current actions, significant deviations from a user's established norms are recognized as anomalous and may indicate misuse/espionage. Most of the existing anomaly and intrusion detection systems are oriented towards detection of anomalous user behavior [3,4]. While a few anomaly detection prototypes have addressed anomalous behavior via host monitoring, discriminating between a users behavior and the effects of malicious software has not been demonstrated to date. To this end we are currently in the midst of an R&D effort to extend the current ISOA prototype to include program/process monitoring capabilities.

A process can be defined as an instance of a program in execution, which can be expected to exhibit a range of predictable behaviors. These behaviors are in part dependent on the execution environment. Analyzing software at the levels of source code, object code, and executable code can reveal increasingly detailed information about expected process behavior. Such analysis can lead to the listing of the system calls, resources, etc. invoked or accessed by the software. A system that "tags" software in this manner, and performs run-time capabilities checking could be implemented as an extension to the operating system.

While analysis of source code will reveal overall functionality that is useful for understanding a piece of software, it is unlikely that any analysis short of monitoring a currently executing program will reveal the true range of behaviors for some software. Unexpected run-time situations (bugs), self modifying code, run-time libraries, and dynamic linking of software modules preclude the exhaustive specification of actual behaviors that will be exhibited by software. While this description represents the extreme case, the possibility of obtaining useful measures of expected behavior has yet to be demonstrated.

We are developing a tiered model for process behavior monitoring. Figure 4 depicts this model as consisting of the following levels:

- Process Capabilities — Real-time process capability checking based on an analysis of the process; in UNIX, this includes permitted system-calls and information about valid file-system resources. The goal of real-time process monitoring is to identify unexpected process behavior.
- Profile Specified Behavior — Performance and usage metrics (similar to user profiling); this level consists of statistical and rule-based descriptions for expected behavior. Monitoring at this level would be "session" based, i.e., at process termination.
- Life-cycle — Information about the process, including: originator(s), modification history, known bugs, security implications of interaction with other specific processes, etc. Thus, program development information would be used as an adjunct to monitoring active processes.

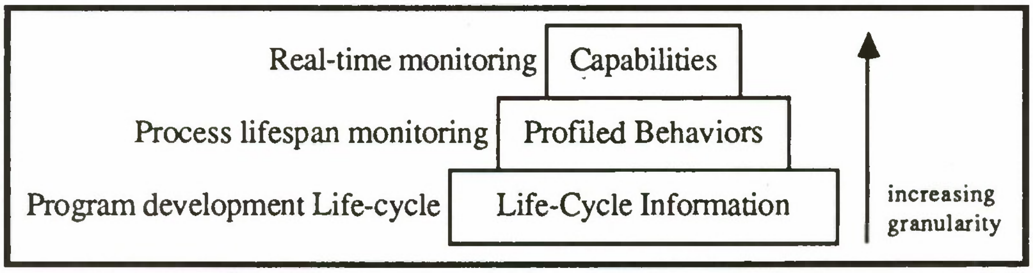


Figure 4. Tiered model for process monitoring

Given these levels, analyzing the behavior of processes requires both collection of information about the program and collection of information to permit process monitoring. Information from any one of these levels would be useful at the other levels. Briefly, a digital record of observed behaviors could be invaluable during software updating/maintenance. While some information available from the development environment would likewise support real-time and session monitoring. Since the performance penalties of low-level auditing of processes are overwhelming, it is unlikely that the capabilities level can be reasonably implemented outside of the operating system kernel.

We currently monitor users and hosts in a UNIX network environment. However, since we convert all audit records into a canonical form it will be relatively simple to monitor non-UNIX hosts by adding the equivalent daemon and audit support as required. Differences in the kinds of auditable events could be easily handled since profiles are currently specified on a per user and host basis.

The goal for the analysis of audit records is the reduction of massive amounts of audit records into a form that is meaningful and readily comprehended. As presented in this paper, the ISOA offers a rich environment for the collection and analysis of audit traffic in networks that require security monitoring. By integrating direct security control of individual user sessions and host operations, the ISO has available the necessary tools for intervention as indicated by the monitoring and analysis of user and host behavior.

Acknowledgements

This paper owes credit to the ideas and collective work of Jody Heaney, Nan Fredman, John Page, Mark Ambrose, and Frank Serrao. In addition, thanks are due to Dr. Jude Franklin and Dr. John White for their continual encouragement.

References

[1] Winkler, J.R. and Page, W.J., "Intrusion and Anomaly Detection in Trusted Systems," *Proceedings of the 5th Annual IEEE Computer Security Applications Conference*, December 1989.

[2] Winkler, J.R. and White, J.S., "Surveillance and Anomaly Detection in Secure Networks," *Proceedings of the AFCEA West Intelligence Symposium,* San Diego, March 1990.

[3] Lunt, T.F., "Automated Audit Trail Analysis and Intrusion Detection: A Survey", *Proceedings of the 11th National Computer Security Conference*, October 1988.

[4] Bauer, D.S and Koblentz, M.E., "NIDX - A Real-Time Intrusion Detection Expert System", Proceedings of the Summer 1988 USENIX Conference, June 1988.

[5] Halme, L. R. and Kahn, B. L. 1988. "Building a Security Monitor with Adaptive User Work Profiles." *Proceedings of the 11th National Computer Security Conference,* October 1988.

[6] Sebring, M. M., Shellhouse, E., Hanna, M. E., and Whitehurst, R. A. 1988. "Expert Systems in Intrusion Detection: A Case Study." *Proceedings of the 11th National Computer Security Conference,* October 1988.

[7] Vaccaro, H.S., and Liepins, G.E., 1989. "Detection of Anomalous Computer Sessions Activity", *Proceedings of the 1989 IEEE Computer Society Symposium on Security and Privacy.*

[8] Anderson, J.P. 1980. "Computer Security Threat Monitoring and Surveillance". James P Anderson Co., Fort Washington, PA, April 1980.

[9] Lunt, T. F., Jagannathan, R., Lee, R., Listgarten, S., Edwards, D. L., Neumann, P. G., Javitz, H. S., and Valdes, A. 1988. *IDES: The Enhanced Prototype, A Real-Time Intrusion-Detection Expert System.* SRI-CSL-88-12. Menlo Park, CA: SRI International, Computer Science Laboratory.

[10] Clyde, A.R., "Insider Threat Identification Systems", *Proceedings of the 10th National Computer Security Conference*, September 1987.

[11] Denning, D., "An Intrusion-Detection Model", *Proceedings of the 1986 IEEE Symposium on Privacy and Security*, April 1985.

A Neural Network Approach Towards Intrusion Detection

Kevin L. Fox   Ronda R. Henning   Jonathan H. Reed
Richard P. Simonian
Harris Corporation
Government Information Systems Division
P.O. Box 98000
Melbourne, FL 32902

July 2, 1990

Abstract

Current generation intrusion detection technology primarily relies on audit trail analysis techniques to determine if an intrusion has occurred. Neural networks afford a flexible pattern recognition capability that can be adapted for intrusion detection purposes. A prototype anomaly detection system using self-organizing feature maps is described, and an architecture for a general intrusion detection system based on this prototype is discussed.

1 Introduction

In this paper we discuss a unique approach towards computer intrusion detection, damage assessment, and removal. The approach is a host-independent monitoring system which uses neural networks to learn and track the system-normal state, coupled with a expert system for in-depth intrusion analysis. The system may also make use of existing static analysis tools for post-incident prevention activities. There are several advantages to this approach:

1. Adaptive modelling of the users and the system.
2. Ability to deal with unknown viruses or intrusions.
3. Determination of when to use the more computationally expensive expert system.

We begin with an analogy between biological and silocon-based infection characteristics, and continue with an examination of the state-of-the-art for intrusion/virus detection mechanisms. We then describe the functionality of the expert system and the neural network in our proposed architecture. We conclude with an outline of the ideas for future research.

2 An Analogy

Computer 'viruses' are aptly named when one considers the similarities between the infection and propagation methods[1] between silicon and biological viruses. However, a biological organism has much better defensive mechanisms against viruses and other infections than computers currently have.

A biological organism defends itself against infections and viruses by producing antibodies. Antibodies are molecules whose chemical and morphological properties enable them to recognize, bond to, and destroy (or at least deactivate) infectious molecules. Antibodies are not general in nature; rather, they are targeted to a specific infection. An organism "learns" a virus through exposure to that virus; thus, vaccinations are meant to introduce a controlled amount of a virus to the organism so that antibodies will be produced against it. The next time that virus appears, the existing antibodies enable the organism to quickly recognize it and respond. Of course, if an organism is affected by a new infection, it may not be able to react in time to prevent the infection from spreading and causing damage.

We feel that certain Artificial Intelligence techniques could be effectively employed to mimic the biological response to viruses and infections. Specifically, an artificially intelligent computer could be made to monitor itself, recognize foreign invaders, and formulate the appropriate defense. Through the integration of Artificial Intelligence and Computer Security, we believe that a dynamic system can be built which can be trained to recognize a virus attack and to take the required action.

3 Existing Systems

Young [15] describes two types of monitors which can be used to recognize viruses and other intrusions: appearance monitors and behavior monitors. Appearance monitors perform a static analysis of computer systems to detect anomalies in source or executable files, such as replicated code. Behavior monitors dynamically examine the behavior of processes for dangerous actions, such as reading a directory or writing to an executable file, or suspicious activity. Both types of monitors could run as background processes, or be interleaved into the operating system.

3.1 Appearance Monitors

Appearance Monitors are generally static analysis tools. There are many proposed methods of examining a system for damage with an appearance monitor. There are **virus killers** which will search for and remove a specific virus. With this approach, one is always playing 'catch-up' with the viruses; when a new virus starts making the rounds, an appropriate virus killer must be developed and distributed.

[1] And, unfortunately, the potential for damage.

There are more general source and object code analysis tools which look for discrepancies such as increased executable image size, common or repeated code in files, and inconsistent coding styles. Garnett [4] proposed a selective disassembly scheme which would look at conditional statements in object code. His claim is that illicit code requires a **trigger**, such as a check for previous infection, or a check for an activation date. Thus, illicit code may be detected by disassembling and examining conditionals.

Static analysis tools have several disadvantages. The analysis is computationally expensive since it needs to examine a major portion (if not all) of the system's files. Analysis must be invoked manually or by some after-the-fact trigger. Someone who knows what these tools look for may be able to subvert the system with a more clever virus. The advantage of such tools is that they can perform a very in-depth analysis of the system and that they can be implemented relatively quickly.

3.2 Behavior Monitors

Some work has been done using statistical analysis to determine if an intrusion is occurring, or to assist in pinpointing the source of an intrusion [5, 8]. In general, these systems identify a set of auditable system parameters, ingest the data for some period of time, and come up with a profile of the system and user 'acceptable' states. Monitors will then, either statically or dynamically, examine a snapshot of the system and take some action if limits have been exceeded.

One example of a real-time monitor is IDES (Intrusion-Detection Expert System) from SRI [3]. IDES is "based on the hypothesis that any exploitation of a computer system's vulnerabilities entails behavior that deviates from previous patterns of use of the system; consequently, intrusions can be detected by observing abnormal patterns of use" [9]. IDES updates its profiles of user activity periodically and has a rule-based expert system to examine abnormalities.

MIDAS (Multics Intrusion Detection and Alerting System) monitors user commands on DOCKMASTER. It uses heuristic rules to identify various types of intrusions, including **Immediate Attack, User Anomalies**, and **System State**. Again, MIDAS maintains user statistical profiles. MIDAS runs in real time, and is slightly oversensitive because of the brittleness of statistical profiling [11].

4 Expert Systems

As seen in previous examples, one artificial intelligence approach to the problem of intrusion detection is the use of expert systems. Rule-based diagnostic systems in particular are one of the most successful types of expert systems. For example, the MYCIN project [12] during the mid-70's developed an expert system for diagnosing and treating blood infections. The process of detecting and treating silicon infections is the same abductive process: generate a hypothesis of the infection type based on available data and knowledge, and suggest a plan for treatment.

Such a system requires knowledge from human experts on intrusion/virus detection and removal. For example, a human expert might recognize not only the direct effects of a virus (aborted processes, wiped-out hard disk), but also subtle side effects of a clever virus (occasional missing files, high CPU utilization). By exploiting this knowledge while monitoring the computer, an expert system monitor could respond to intrusions more quickly and accurately. Further, the rule-based form of knowledge in an expert system simplifies the process of modifying and extending the system to recognize new threats.

A problem with expert systems is that they are computationally expensive. An expert system with a reasonably large rule base could not feasibly run as a background monitor without degrading system performance. The MIDAS project has installed their expert system on a Symbolics computer which obtains Multics process information via a download procedure. This approach does not degrade the main computer's performance, but it does require the maintenance of a separate and expensive Lisp machine. In our proposed architecture, the expert system would reside on the host computer and would be invoked only when necessary.

Although rules in an expert system are easy to add, delete, and modify, the rule base also clearly defines the situations that the system can react to. It would be very difficult, if not impossible, to implement an expert system which is general enough to recognize and respond to any sort of virus or intrusion. Such a rule base would be infeasible, both from a development and execution standpoint. We believe that a neural network would provide an efficient and elegant front-end status monitor which is also general enough to recognize unknown viruses and possible malicious user behavior patterns.

5 Artificial Neural Networks

Neural networks are a model of computation that roughly models biological neural connections in the brain. This approach is radically different from the traditional sequential models because it is composed of many highly parallel nonlinear computational nodes.

In an artificial neural network, information representation occurs as connection weights between processing elements in the network, and information processing consists of the elements transforming their input into some output as modulated by the weights of connections to other units[2].

5.1 General Architecture

Neural networks are constructed of many small computing elements and connections between the elements. Each node has a simple state associated with it and, depending on the neural network, some algorithm or heuristic for updating the state. *Weights,* or *strengths,* are associated with the input connections of each node. This construction is patterned after biological neurons and synapses. It is believed that biological memory is stored in the weights between

[2]Lippman's "An Introduction to Computing with Neural Nets," [7], is an excellent introductory article on neural networks for those interested in learning more.

neurons. A pattern will trigger a memory in a biological system because the strengths among a set of neurons have been increased to respond to that pattern. This is the same process artificial neural networks use.

Neural networks can be implemented which learn patterns over time. Generally, these models will use activation rules which compute a new value based on the old value as well as on the set of inputs. Thus, new states are functions of experience.

Biological neural networks are constructed of neurons and synapses, with acetocholine controlling the connection strengths. Artificial neural networks may be simulated in software, or built from "simple electronic components: operational amplifiers replace the neurons, and wires, resistors, and capacitors replace the synaptic connections. The output voltage of the amplifier represents the activity of the model neuron, and currents through the wires and resistors represent the flow of information in the network" [14].

5.2 Self-Organizing Neural Networks

The *Self-Organizing Feature Maps* of Kohonen belong to that class of artificial neural network classifier which is unsupervised during learning. (See Table 1, taken from Lippman, [7].) This network differs from the more familiar *Perceptron* and the *Multi-layer Perceptron* which learn via supervised training.

Neural Net Classifiers For Fixed Patterns					
Binary Input			Continuous-Valued Input		
Supervised		Unsupervised	Supervised		Unsupervised
Hopfield Net	Hamming Net	Carpenter/ Grossberg Classifier (ART)	Perceptron	Multi-layer Perceptron	Kohonen's Self-Organizing Feature Map

Table 1: A taxonomy of six neural networks that can be used as classifiers.

The *Self-Organizing Feature Map (SOFM)* networks consist of a single layer of neurons, referred to interchangeably as neurons, processing units, nodes, etc. Each processing unit in the SOFM network is specifically matched or sensitive to a particular domain of input signals in a regular order. These networks represent knowledge from a particular domain in the form of a Feature Map that is geometrically organized. This organization is achieved during the training without supervision by the use of lateral feedback, thus providing a general collective phenomena.

5.3 Applications

We have identified two potential uses for Neural Networks in an intrusion detection application. The first use would be to learn specific virus patterns and to take some action if that virus (or a similar mutation) appeared. The second use would be to adaptively model the normal state for users and the system, and take some action when any abnormality is noted.

5.3.1 Specific Viruses

Biological antibodies essentially perform pattern matching against viruses. If a reasonable taxonomy of viruses can be developed, an artificial neural network could be trained to recognize them. Neural networks are ideal for fast, parallel pattern recognition and for adaptive learning. The use of a neural network would allow the computer to be "vaccinated" against viruses. The network would be trained by introducing samples of existing viruses to the system. Their patterns would be learned and associated with a human-prescribed antidote in each case. The next time the pattern appears in the system, the neural network monitor would trigger (or suggest) the defense.

A neural network could be trained to recognize a wide variety of virus patterns, from mail messages beginning with an "X", to sustained high CPU utilization. The advantage of a neural network is that if a new virus appears which the computer hasn't been vaccinated against, the network should still be able to recognize it as suspicious activity and notify the operator. At the same time, it would be able to learn the new pattern for future use.

5.3.2 Modelling System and User Normalcy

We believe that a more efficient and powerful use of neural networks is to adaptively model system and user normal state. Other systems, such as MIDAS and IDES, perform this modelling through statistical analysis of audit data. Our work in neural networks and a prototype of our ideas applied to a distributed system architecture have convinced us that Kohonen Self Organizing Feature Maps are ideally suited for this task.

One of the advantages offered by the use of the *Self-Organizing Feature Maps* is that while an appropriate (or comprehensive) list of system parameters for monitoring by the network is required, it is not necessary that the features be weighted. The network can learn relationships between features by learning similarities according to some user-defined metric.

6 Proposed Architecture

In this section we elaborate on the system architecture alluded to in previous sections. The key concept is that a Kohonen Self Organizing Feature Map will be used as a real-time background monitor to adaptively model system and user normalcy. When deviations occur, an operator can be notified who may choose to invoke the expert system to perform a more in-depth analysis of the possible problem. The expert system, in turn, may make use of other static analysis tools. As soon as the neural net notices a deviation, it may be configured to notify the operator, log a report, or take a more drastic preventive measure such as temporarily freezing all processes while static analysis proceeds. Figure 1 shows our proposed system architecture.

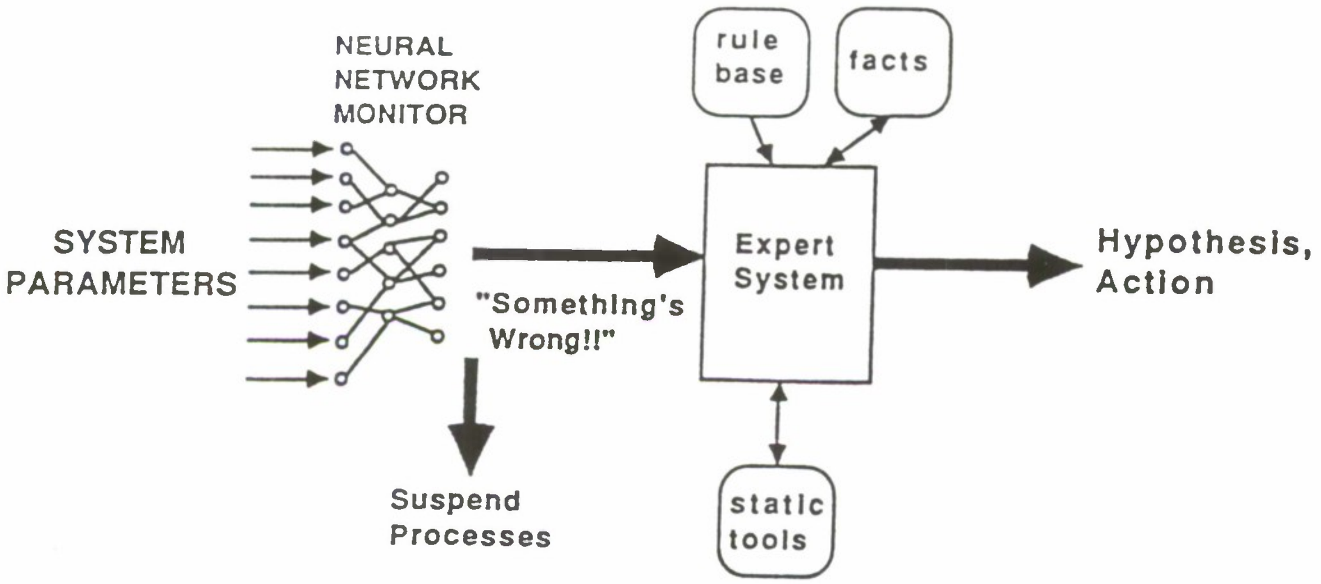


Figure 1: **Proposed System Architecture**

The purpose of the neural net is to learn the normal system activity and adapt to gradual changes. Rapid changes would trigger invocation of an expert system. The expert system's purposes would be to:

- Verify the intrusion, perhaps with other static analysis tools.
- Classify the virus or attack type.
- Suggest a defense, or automatically employ the defense.
- Provide an explanation facility for the operator.

The expert system component would be able to draw upon previous work in this field, including the IDES and MIDAS systems.

Once again, the advantages this system would have over existing intrusion detection systems are: efficiency; the ability to adaptively model both specific users and the system as a whole; the ability to deal with unknown viruses; and the integration of detailed expert knowledge.

7 A Prototype

We have prototyped the neural network portion of our architecture to demonstrate its applicability to current generation computer system architectures. We identified a set of eleven system parameters which are accessible from the system statistical performance data and are also likely to change during an intrusion attempt. These parameters are:

1. CPU Utilization
2. Paging Activity
3. Mailer Activity
4. Disk Accesses
5. Memory Utilization
6. Average Session Time
7. Number of Users
8. Absentee Jobs
9. Reads of "Help" Files
10. Failed Logins
11. Multiple Logins

For our initial prototyping efforts, appropriate statistical simulations for each parameter (for a pseudo-VAX machine) were developed. The models were tri-modal, with peaks at mid-morning, mid-afternoon, and at midnight. We set up this input vector on a SOFM tool developed internally on a Symbolics computer. After some initial experimentation, a SOFM network with 144 nodes (arranged in a 12 x 12 array) was selected. All nodes in the network were initialized with with weight vectors $0 \in \Re^{11}$. The network was trained using Kohonen's learning algorithm [6] with model parameter data for four days, with samples drawn every minute. After the completion of the learning phase, the network was run in a classification mode on data for one day. At approximately 10:10 am, a simulated virus attack was launched. It ended at approximately 10:55 am.

One aspect of the prototype is a graphical representation of the input's deviation from 'normalcy'. The upper left-hand corner of the screen contains a window labeled *Distance*. In this window we plot a moving average of the distance from the input vector and the weight vector of the node which was classified as the winning node. Prior to an attack, the plot of the distance is relatively flat. As an intrusion progresses, the distance graph increases sharply. When the intrusion subsides, the distance graph will decrease to illustrate 'normal' levels of activity. The *Feature Map* window displays a 12 x 12 network of nodes, with the number displayed at each node representing the frequency of a particular node being the winning node during learning. Additional windows allow user interaction with the SOFM network during its learning phases, and permit monitoring of its operation when the network is used autonomously.

8 Results of the Prototype

The neural network monitor simulation worked as expected and was successful in detecting suspicious activity in a general purpose user environment. Future plans for our prototype activity include:

- Distillation of the monitoring code to its minimum configuration. The neural network simulator used for the prototype is more robust than our intrusion detection monitor requires, and subsequently is not tuned to our application in CPU utilization or memory constraints.

- Implementing the prototype monitor in a multi-user system to determine its impact on system performance.
- Developing the rule base for subsequent attack diagnosis.
- Exploring use of the architecture for network monitoring/management in distributed environments.
- Applying the architecture to operational systems.

The neural network approach is not without its drawbacks. A network that can self-organize may, in time, be subject to a very subtle attack without recognizing that an attack is occurring. In this scenario, an intruder would take actions slightly out of tolerance with a system's normal behavior over a period of time. Such gradual changes may not be detectable by the monitor unless it is also being monitored by a less tolerant neural network.

When connected with network management functions in a distributed environment, the propagation rate of the infestation or intrusion may make the monitor's notification of abnormal activity too late for the system security officer to prevent subsequent infection. One solution to this problem would be to remove a suspicious node from the network immediately upon suspicion of attack, make a short, preliminary assessment, and then determine if further investigation is warranted prior to reconnection.

9 Conclusions

The self-organizing feature map has provided a basis for our preliminary work in neural network based intrusion detection techniques. Early results indicate that this architecture is most promising, and our future research is concentrating on refining the neural network for unobtrusive background monitoring.

References

[1] B. Chandrasekaran, A. Goel, D. Allemang, "Connectionism and Information-Processing Abstractions", *AI Magazine*, Winter 1988, Vol. 9, No. 4.

[2] Russell Davis, "Exploring Computer Viruses", *Proceedings of the 4th Aerospace Computer Security Applications Conference*, Orlando, Fl, December 1988.

[3] D. E. Denning, P. G. Neumann, "Requirements and Model for IDES - a Real-Time Intrusion Detection System", Computer Science Laboratory, SRI International, 1985.

[4] Paul Garnett, "Selective Disassembly: A First Step Towards Developing a Virus Filter", *Proceedings of the 4th Aerospace Computer Security Applications Conference*, Orlando, Fl, December 1988.

[5] H. S. Javitz, A. Valdes, D. E. Denning, P. G. Neumann, "Analytical Techniques Development for a Statistical Intrusion Detection System (SIDS) based on Accounting Records", SRI International, Menlo Park, CA, July 1986.

[6] Teuvo Kohonen, "Tutorial Number 10: Self-Organizing Feature Maps", IEEE International Conference on Neural Networks, San Diego, CA, 1988.

[7] Richard P. Lippman, "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine*, April, 1987.

[8] Teresa Lunt, J. van Horne, L. Jalme, "Automated Analysis of Computer System Audit Trails", *Proceedings of the 9th DOE Computer Security Group Conference*, May 1986.

[9] Teresa Lunt, "Automated Audit Trail Analysis and Intrusion Detection: A Survey", *The 11th National Computer Security Conference Proceedings*, October 1988.

[10] David E. Rumelhart, J. L. McClelland, Parallel Distributed Processing, Vol. 1 and 2, The MIT Press, Cambridge, Massachusetts, 1986.

[11] Michael M. Sebring, E. Shellhouse, M. Hanna, "Expert Systems in Intrusion Detection: A Case Study", *The 11th National Computer Security Conference Proceedings*, October 1988.

[12] E. H. Shortliffe, Computer Based Medical Consultations: MYCIN, American Elsevier, New York, 1976.

[13] Richard Simonian, "Applying Neural Networks to Expert Systems", Harris Internal Report AI-TR-88-14.

[14] David Tank, J. L. Hopfield, "Collective Computation in Neuronlike Circuits", *Scientific American*, December 1987, pp. 104-114.

[15] Catherine Young, "Taxonomy of Computer Virus Defense Mechanisms", *The 10th National Computer Security Conference Proceedings*, September 1987.

A GENERALIZED FRAMEWORK FOR ACCESS CONTROL: AN INFORMAL DESCRIPTION

Marshall D. Abrams * Kenneth W. Eggers *
Leonard J. La Padula † Ingrid M. Olson *

The MITRE Corporation
* 7525 Colshire Drive, Mc Lean, VA 22102
† Burlington Road, Bedford, MA 01730

ABSTRACT

This paper introduces a framework for studying and constructing access control policies for automated information systems. This framework provides a view of access control policies as *rules* specified in terms of *access control information* and *context* by *authorities*.

- Access Control Information (ACI) — Characteristics or properties of subjects and objects. Their names are used in specifying the rules of the system; their values are used by the access control rules.
- Access Control Context (ACC) — Additional information, such as time of day, used in access control decision making.
- Access Control Authorities (ACA) — Agents who specify ACI, ACC, and rules.
- Access Control Rules (ACR) — The set of formal expressions of policy for adjudicating requests by subjects for access to objects.

These four factors cover the key choices and constraints for the designer of a system. All of the potential policies we have examined can be expressed in their terms.

INTRODUCTION

The thesis of this paper is that a more general, uniform approach to access control in Automated Information Systems (AIS) can lead to trusted systems of greater utility. Traditional access control policies, such as Mandatory Access Control (MAC) and Discretionary Access Control (DAC)[1], are merely two possible points in a broad space of access control policies. This paper provides a general, informal description of an approach for constructing access control policies that can be used to satisfy a wide range of complex security policies. This paper incorporates prior work [1] updated by continuing research.

While studying existing access control policies (such as DAC and MAC), several proposed modifications and enhancements to these policies, and other proposed access control policy models (such as the Clark-Wilson integrity model [4]), a framework for a uniform approach to access control took shape, which we have named the Generalized Framework for Access Control (GFAC). Using this framework to examine the similarities, differences, strengths, and weaknesses of existing policies, we derived general concepts that may improve existing policy models and create new policy models, leading to improved access control mechanisms. GFAC includes MAC and DAC as specific designs which can be implemented by choosing the appropriate design parameters. Existing systems, their models and evaluations, are not affected by GFAC — except, perhaps, as to how we think about them. We believe that a major contribution of GFAC is a change in emphasis and viewpoint. Like those programming languages

This paper was supported by the National Computer Security Center under contract F19628-89-C-0001 and by The MITRE Corporation. The opinions expressed do not necessarily represent the position of either organization.

[1] In this paper, MAC and DAC are treated as reserved words referring to Mandatory Access Control and Discretionary Access Control respectively, as defined in [12].

that try to reduce the probability of programmer error by providing an environment that encourages some practices and discourages others, GFAC provides a framework that encourages explicit inclusion of desired security functionality in the rules.

Schaefer [16] and Landwehr [8] have previously commented on the use of trusted subjects and processes in order to overcome some of the overly restrictive axioms of the models (e.g., Bell-LaPadula (BLP) Model [2, 3]) used for secure system development. These trusted subjects and processes are endowed with special exemptions from some or all of the policy enforcement by the reference validation mechanism or other parts of the Trusted Computing Base (TCB). These exemptions are necessary for the trusted subjects to perform their intended functions. When the policy enforced by the trusted subject is different from the policy described in the system security model, the validity of the model as a representation of the system is compromised and assurances derived from formal analysis of the model are rendered invalid. By directly addressing the policies associated with these trusted subjects and processes in the formal model and specifications, no exceptions or special cases are necessary.

Organization

The next section presents GFAC in terms of its fundamental components, using DAC and MAC as examples to illustrate concepts. Then we discuss two applications of GFAC — the Clark-Wilson integrity model, and handling restrictions used in the DOD/intelligence community. We close with some comments on continuing and future GFAC research.

COMPONENTS OF THE GENERALIZED FRAMEWORK FOR ACCESS CONTROL

The main premise of GFAC is that all access control is rule-based. This idea has been suggested elsewhere in various forms. [7, 14, 15, 18] GFAC is consistent with the framework for access control in open systems being developed in the standards community [6], and adopts the terminology from that work.

There are four principal components used in the implementation of access control, covering the key choices and constraints for the designer of a system — access control information (ACI), access control context (ACC), access control rules (ACR), and access control authority (ACA). Each component is discussed in more detail in the following sections.

Access Control Information (ACI)

ACI is associated with subjects and objects; it reflects their characteristics and security attributes. The names of ACI items are used in specifying the security rules of a system; the values of these items are used by the rules to determine whether a given subject may access a specific object. A set of named ACI items is associated with a class of subjects or objects and a particular access control policy.

ACI related to subjects might include identification data (e.g., user ID, name, employee number), authentication data (e.g. password, smart card PIN, fingerprint), biographic data (e.g., department, nationality), clearance, location, access permissions relative to classes of objects (e.g., capabilities), and role (e.g , user, system administrator, security officer).

ACI related to objects might include classification, handling restrictions (e.g., EYES ONLY, CLOSE HOLD), classification authority, source/originator, document number, owner, a list of programs allowed to access the object and their access permissions (e.g., Clark-Wilson model to enforce the well-formed transaction), and identities of users and their access permissions (e.g., access control list).

Access Control Context

The ACC contains information not associated with an subject or object but necessary to the access control decision process. The information becomes security relevant by virtue of being used by ACR. The integrity of this context information must be protected by preventing unauthorized changes. Security policy may also require secrecy protection.

The access control context might include time — access to the information (i.e., sensitivity of the information) may vary with time (e.g., the Department of Labor Statistics information on last month's unemployment rate is sensitive until 9:00 am on Tuesday morning when it is made public), status — the access control restrictions depend on a status variable which is officially changed to reflect some condition in the real world (e.g., crisis or exercise status), and group membership — the names of groups are ACI associated with objects, but the definition or enumeration of membership in a group is part of the context.

The ACC represents aspects of the physical and logical environment, including status variables representing the condition of the real world (e.g., whether there is a real crisis or a practice exercise is in progress) as well as information representing the state of the AIS. Although context information can be regarded as another kind of access control information, ACI and ACC are differentiated by their association with subjects and objects. ACI is associated with subjects or objects; ACC is not.

Access Control Rules (ACR)

Access control rules (ACR) are the regulating principles that define the access control policy. In a trusted AIS, access to an object by a subject is controlled by a TCB. The TCB will often provide some set of system functions (e.g., open file, activate process, delete file) as its interface to user processes. As a part of the normal operation of these functions, they also adjudicate the request for access according to built-in security policy rules. These system functions are sometimes referred to as the security or access control "rules" of the system; however, this terminological convention tends to be confusing.[2] In this paper, we use the term *rule* to identify only the portion of the function that adjudicates the access control requests. That is, we separate the system function into two operations: one in which adjudication of the access is requested (i.e., the ACR are invoked) from some TCB-resident security "rule-base," and a second that performs the requested non-policy-related functions (e.g., establishing access between a subject and a file, initializing a new subject, or removing a file object from the system). In this view, the rule-base adjudicates requests according to the following general principle:

A subject is permitted to access an object in access mode M only if the ACI of the subject, the ACI of the object, and the current state of the ACC satisfy the rules.

Combination of Rules

Rules implementing multiple security policies must reflect how these policies relate to each other. For example, in combining MAC and DAC, neither MAC nor DAC takes precedence with respect to denying access. The MAC and DAC decisions are logically ANDed together; either decision process may deny access. Since MAC and DAC decisions are usually implemented to operate sequentially, their temporal sequencing is sometimes mistaken for precedence. Another form of combination occurs when there are multiple conditions for adjudicating access. For example, consider conditions A and B. It is a business decision whether access should be granted based on A AND B or A OR B.

Precedence does exist in the Trusted Computer System Evaluation Criteria (TCSEC) [12], which requires for DAC (at class B3) the ability to specify (for a given object) specific users and groups and their respective modes of access to the object, including no access. This policy can lead to a number of possible interpretations, as discussed in [10]. Briefly, in one interpretation, if an individual user is specifically granted or denied authorization for an object, this takes precedence over any authorizations for the object that are granted or denied in groups to which the user belongs. In another interpretation, denials take precedence; that is, user or group's

[2] The "rules" described in the Multics interpretation of the Bell-LaPadula model [3] can be seen to include some of the non-policy-related functionality described above.

denial of authorization for an object takes precedence over any authorizations that the user or group may have been granted for the object.

Inheritance Rules

An important concept in creating new subjects and objects is "inheritance" [13]. A new object may be simply a copy of an old object, may be created anew, may be created by changing or editing an existing object, or may be formed by combining two or more existing objects. Inheritance rules are concerned with establishing the ACI associated with the new object. The MAC inheritance rule may be inferred from the TCSEC in a rather straightforward manner: a new object is labeled at the sensitivity level at which the user is operating, usually the level at which he or she logged in.

There is no DAC inheritance rule in the TCSEC, a consequence of the discretionary nature of DAC policy. Some implementations, however, provide defaults. UNIX®, for example, allows a user to specify default user/group/world (UGW) protection for all new objects and a copy of an object inherits the old object's ACI when they have the same owner and inherits the owner's ACI otherwise.

Configuring A System's Rules

In principle, GFAC gives the person configuring an AIS's security controls the freedom to specify any rules desired. In practice, the ability to configure security controls is extremely limited in today's trusted systems. Once a system has been evaluated there can be no significant changes made to the configuration under which it was evaluated. It is, of course, a goal of this effort to bring this kind of flexibility to trusted systems. Some work, in a complementary vein, has been done in this area; see [14] on security rule bases.

Our vision is that vendors will provide sets of rules suitable for market segments. For example, rule sets may be produced for DOD, civil government, message system, office automation system, and commercial environments. In this scenario, the system security architect would pick the rule set from the catalog and initialize variables to implement the organization's policy.

An organization with a unique policy, however, might be forced to add or modify rules. Our current research is addressing the question of how such a change in rules would impact the formal assurance of the system. On inspection it appears that any change in the rules should be approached with considerable caution. Manual or automated examination for completeness, consistency, and functionality appears warranted.

Authority

One may associate the notion of span of authority with originators, owners, Information System Security Officer (ISSOs), commands/agencies, and national or corporate policy. In general, higher authority levels will be responsible for establishing the policy, information system architects will translate the policy into rules, the system modelers/designers will represent these rules in a formal manner and will design their implementation, and the ISSO will be responsible for entering and maintaining the subject/object ACI. The structure of the authority for a given system will be determined in part by the rules established for that system, affecting, for example, whether the ISSO is allowed to delegate some of his authority to owners and the exact form and extent of that delegation.

It could be argued that authority considerations are just a subset of the rules, i.e. those rules governing who has the right to change rules and ACI. This may be correct in a formal sense. However, authority has been inadequately addressed in the past and is of fundamental importance equal to the other principal components of GFAC. GFAC makes a significant contribution toward the ability to judge the quality of a real system's policy by explicitly recognizing the rules for authority.

® UNIX is a registered trademark of AT&T.

Control of Access to the Access Control Information

Control of access to the ACI is essential. Controlling the ability to read and modify ACI is key to the strength of a trusted system's access controls. One can organize ACI into sets of attributes with access control authority trees attached to selected attributes. These trees define the authority and privileges in the system. Three levels of hierarchy appear reasonable, although one can imagine more or fewer levels depending on the needs of a particular situation.

Figure 1 **Partial Authority Tree**

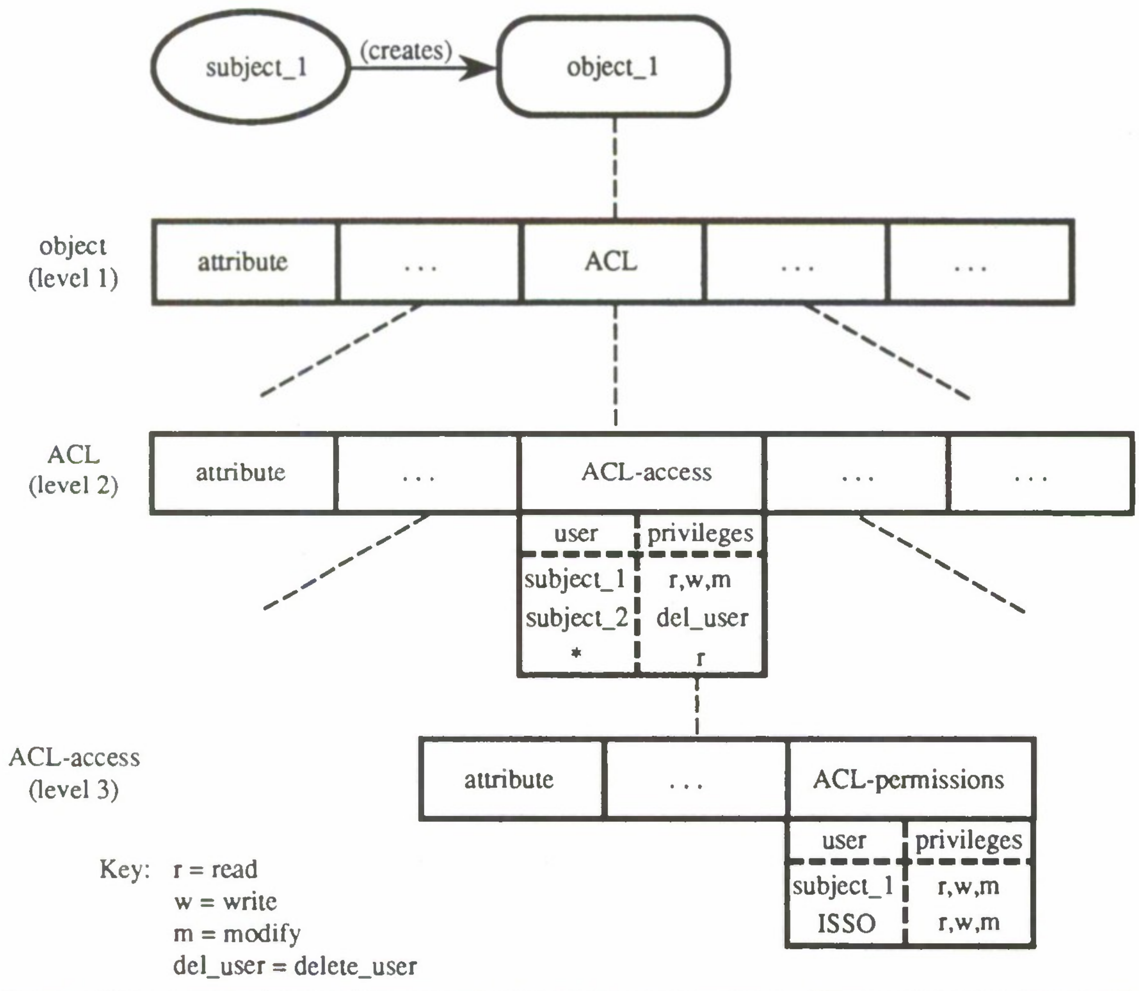


Figure 1 shows an example of a partial ACA tree. Assume that when Subject_1 creates Object_1, the three levels of ACI are created as shown in the figure, except that Subject_1 is the only entry in the ACL-access (level 2) attribute at this time. Assume that the policy includes the concept of owner of an object, and creates an object with the owner having read, write, and modify access permission on all three levels. Similar access control trees may be associated with some or all of the other attributes as well.

Let us examine the meaning of the three levels. At level 1, the object ACI includes the access control list on Object_1 as one attribute. Figure omits the contents of the ACL; let us assume that the policy puts Subject_1 on this ACL. Additional entries may be made on this ACL. But who is allowed to access the ACL? GFAC treats the ACL, an attribute of Object_1, as a specific object. All objects have ACI associated with them. In this example we are interested in the ACL ACI (level 2), to which is attached the ACL-access ACI (level 3). Under the assumed policy, level 2 is initialized granting Subject_1 read, write, and modify privileges on the ACL. Subject_1, choosing to share one of its privileges with Subject_2, enters Subject_2 in the level 2 ACI with the privilege of deleting users from the ACL. Further, Subject_1 gives everyone the privilege to read the ACL.

But what controls access to the level 2 ACI? The assumed policy includes the creation of one more level of ACI, level 3 ACL-access ACI. Level 3 is initialized granting Subject_1 the ability to read, write, and modify the ACL-access attribute. The ISSO is also given read, write, and modify permission, since, in this example, the ISSO is viewed as the ultimate authority within the AIS.

APPLYING THE GENERALIZED FRAMEWORK FOR ACCESS CONTROL

The GFAC view of trusted systems emphasizes four factors in the design of access controls: access control information, context information, rules, and authority. We believe these factors encompass what is needed to define many useful access control policies. This section demonstrates this thesis by discussing 1) a commercial integrity policy and 2) polices for applying dissemination and handling controls common in the intelligence community.

Clark-Wilson Integrity Model

The Clark-Wilson integrity policy [4] is a fairly recent policy introduced as one model of what integrity means to the commercial data processing world. It centers on two main concepts for maintaining integrity: the well-formed transaction and separation of duty, both modeled after well established practices from the general accounting world.

Clark-Wilson Integrity (CWI) provides for both external and internal consistency of data. Measures for external consistency, such as their Integrity Verification Procedures (IVPs), ensure that the data stored in the computer system correctly models the state of the real-world systems it relates to. The IVPs reflect generally accepted audit practices in general accounting. Measures for internal consistency, the well-formed transactions, called Transformation Procedures (TPs) in their model, ensure that data in a valid state is modified in such a way that the resulting state of the data is again valid. The TPs embody accepted practices like double entry bookkeeping. Separation of duty is also reflected in their integrity rules: An agent that can certify an entity (e.g., determine that a TP is correctly implemented) may not have any execute rights with respect to that entity (i.e., is not allowed to run the TP program as a user of the system).

The data that are integrity-controlled under CWI are called Constrained Data Items (CDIs). A CDI, likely to be realized as a file on most computer systems, is validated by an IVP to ensure that the values of the data items in the CDI are in a correct state. This would be done when the CDI is first created and periodically thereafter to ensure that the data corresponds correctly to the real-world aspects of the application of which it is a part. Transactions against a CDI may be performed only by specified TPs and TPs may be operated only by authorized users.

Thus, CWI policy within the computer system is based on

- integrity-controlled programs called Transformation Procedures (TPs) and Integrity Verification Procedures (IVPs)
- integrity-controlled objects called Constrained Data Items (CDIs)
- user permissions to apply certain TPs to specified CDIs.

These computer controls are clear candidates for GFAC implementation, involving significant use of integrity roles, rules, and authorizations.

Handling Restrictions

In the paper world of classified documents within the DOD/intelligence community, numerous dissemination and handling restrictions are applied to documents. Examples include NOFORN (No Foreign Nationals), ORCON (Originator Controlled), and REL XX (Release to nationals from country XX). Williams and Day [17] give an excellent discussion of the complexities of such markings for classified documents, and the inadequacies of current automated systems in handling them. Graubart [5] and McCollum [11] each present detailed arguments demonstrating why DAC, hierarchical MAC, and MAC categories are inappropriate and inadequate for handling ORCON; their arguments apply to other markings as well.

Some efforts have attempted to incorporate the handling of markings into a trusted system. MITRE's CMW prototype [19], based on security requirements of the intelligence community, includes the capability to provide markings in an "information label" that is separate from the MAC sensitivity label. Thus, the CMW prototype includes a labeling policy in addition to the usual MAC and DAC policies, providing a real-world demonstration of the GFAC claim to that effect.

Under GFAC, the appropriate markings and other supporting information needed to make the access control decision would be included as subject/object ACI or additional context information. The implementation of the needed access controls is conceptually straightforward under GFAC. Just as a traditional MAC policy based on a lattice of sensitivity levels can be viewed as a MAC rule (ACR) that uses the sensitivity levels of the subject and object (ACI) to adjudicate access requests, so an extended MAC policy based on a set of markings in addition to the lattice can be viewed as a set of MAC rules (ACR) that use sensitivity levels and markings as well as other subject ACI, like nationality and affiliation, to adjudicate requests. Note that the strength or universal applicability of access control rules is independent of the information on which the rules base their decisions. Thus, the implementation of a labeling policy can be just as strong and pervasive in a trusted system as is the implementation of a traditional MAC policy.

THE NEXT STEP: FORMAL MODELING AND PROTOTYPING

We are taking two directions in our continuing GFAC effort — formal modeling and prototyping. Through formal modeling, the concepts of GFAC will be made more precise; through prototyping, the concepts of GFAC will be made more tangible.

Formal Modeling

One of the main objectives of the GFAC vision is the ability to produce trusted systems in which it is possible to configure the security policy of the system to meet the particular needs of the owners/operators and users. A principal motivation here is the conviction that current trusted systems do not adequately implement the various security policies that people managing documents and other forms of information use and enforce. Two issues, then, for formal modeling are:

- Can we model a useful policy that current trusted systems do not implement and prove that, at least according to an appropriate interpretation of the TCSEC, the resulting system is secure?
- Can we model in a way that will allow configurable security policies for an AIS without having to do a formal evaluation of the AIS for each configuration of policy?

Our preliminary work [9] models system functions like open, read, and write as a policy-free reference validation mechanism. This set of functions appeals to a rule base that expresses access control policies for for the AIS system. The purpose of [9] was to develop a structure for a formal model, with special attention to the form and use of access control rules to support the goals of the GFAC vision.

Our formal modeling approach shows promise of addressing some of the issues that critics of the Bell-LaPadula (BLP) models have raised over the years, specifically the fact that much of the security policy of the system according to BLP is embedded in the system functions. Our formalism uses a separate rule base that explicitly expresses the security policies for the system. Our results, while addressing some issues not dealt with by BLP, do ***not*** suggest that BLP is invalid.

Prototyping

To provide a tangible proof-of-concept, we plan to prototype GFAC concepts. It seems both prudent and efficient to use a system that already provides a B-level MAC policy. Thus, we plan to modify a preexisting TCB to implement several additional policies. Many of the mechanisms used to implement conventional sensitivity labels might carry over, or at least provide inspiration, to the handling of the ACI for these policies. The rest of the TCB outside the kernel (i.e., the implementation of the reference monitor) should be directly useful. AT&T System V/MLS [13] has been selected.

SUMMARY

This paper is a snapshot of our thinking about GFAC. We are continuing with the work. Our thinking has already changed since our first publication [1]. We expect that it will change further as the work progresses. Another version of this paper with more details and examples is available from the first author.

We have only scratched the surface of GFAC, integrating earlier concepts of access control into a general framework. Generalized Framework for Access Control identifies four components — Access Control Information (ACI), Access Control Context (ACC), Access Control Rules (ACR), and Access Control Authority (ACA) — as the key factors in the design of access controls. By making design decisions about each of these variables and their combinations, alternative access control policies can be implemented. GFAC provides an improved framework for expressing and integrating multiple policy components. Associating access controls with an explicit inheritance policy opens up many possibilities for enforcing additional policies.

The simplicity and symmetry of the concept of GFAC is encouraging and indicates that further work is warranted. The correspondence of the ideas expressed in this paper with prior work further reinforces this belief. GFAC continues the mainstream of access control, extending concepts from prior work in a logical evolutionary manner.

BIBLIOGRAPHY

1. Abrams, M.D., A.B. Jeng, and I.M. Olson, *Generalized Framework for Access Control: An Informal Description,* MTR-89W00230, The MITRE Corporation, September 1989. (Also available through National Technical Information Service (NTIS), Springfield, VA, No. PB90 161977/AS).

2. Bell, D.E. and L.J. LaPadula, *Secure Computer Systems: Mathematical Foundations,* ESD-TR-73-278, Vol. I, The MITRE Corporation, March 1973.

   Bell, D.E. and L.J. LaPadula, *Secure Computer Systems: A Mathematical Model,* ESD-TR-73-278, Vol. II, The MITRE Corporation, May 1973.

   Bell, D.E. and L.J. LaPadula, *Secure Computer Systems: A Refinement of the Mathematical Model,* ESD-TR-73-278, Vol. III, The MITRE Corporation, December 1973. (Vol. I-III are also available through National Technical Information Service, Springfield, VA., NTIS AD-780528.)

3. Bell, D.E. and L.J. LaPadula, *Secure Computer Systems: Generalized Framework for Exposition and Multics Interpretation,* ESD-TR-75-306, The MITRE Corporation, July 1975.

(Also available though National Technical Information Service, Springfield, VA, NTIS AD-A023588.)

4. Clark, D.D. and D.R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," *Proceedings of the 1987 Symposium on Security and Privacy*, Oakland, CA, IEEE Computer Society Press, April 1987, pp. 184-194.
5. Graubart, R.D., "On the Need for a Third Form of Access Control," *Proceedings of the 12th National Computer Security Conference*, Baltimore, MD, 10-13 October 1989, pp. 296-304.
6. International Standards Organization, International Electrotechnical Committee, Joint Technical Committee 1, Subcommittee 21, *Working Draft on Access Control Framework*, document number 4206, December 1989.
7. Kurzban, Stan, "Toward A Model for Commercial Access Control," Integrity Workshop, National Institute of Standards and Technology, January 1989.
8. Landwehr, C., C. Heitmeyer, and J. McLean, "A Security Model for Military Message Systems," *ACM Transactions on Computer Systems,* Vol. 2, No. 3, August 1984, pp. 198-222.
9. LaPadula, L.J., "Formal Modeling in a Generalized Framework for Access Control," *Proceedings of the Computer Security Foundation Workshop III,* 12 June 1990, pp. 100-109.
10. Lunt, T. F., "Access Control Policies: Some Unanswered Questions," *Computers & Security,* Vol. 8, 1989, pp. 43-54.
11. McCollum, C.J., J.R. Messing, and L. Notargiacomo, "Beyond the Pale of MAC and DAC — Defining New Forms of Access Control," *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, Oakland, CA, IEEE Computer Society Press, May 1990.
12. National Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria,* DOD 5200.28-STD, December 1985.
13. National Computer Security Center, *Final Evaluation Report of American Telephone and Telegraph System V/MLS Release 1.1.2 Running on UNIX System V Release 3.1.1,* CSC-EPL-89/003, 18 October 1989.
14. Page, John, Jody Heaney, Marc Adkins and Gary Dolsen, "Evaluation of Security Model Rule Bases," *Proceedings of the 12th National Computer Security Conference,* 10-13 October 1989, pp. 98-111.
15. Sandhu, Ravi, "Transaction Control Expressions for Separation of Duties," *Proceedings of the 4th Aerospace Computer Security Applications Conference*, Orlando, FL, December 1988, pp. 282-286.
16. Schaefer, Marvin, "Symbol Security Condition Considered Harmful," *Proceedings of the 1989 Symposium on Security and Privacy*, Oakland, CA, IEEE Computer Society Press, May 1989, pp. 20-46.
17. Williams, J.C. and M.L. Day, "Sensitivity Labels and Security Profiles," *Proceedings of the 11th National Computer Security Conference,* 17-20 October 1988, pp. 257-266.
18. Wood, C., E.B. Fernandez, and R.C. Summers, "Database Security: Requirements, Policies, and Models," *IBM Systems Journal*, Vol. 19, No. 2, 1980.
19. Woodward, J.P.L., "Exploiting the Dual Nature of Sensitivity Labels," *Proceedings of the 1987 Symposium on Security and Privacy,* Oakland, CA IEEE Computer Society Press, April 1987, pp. 23-30.

Automated Extensibility in THETA*

Joseph R. McEnerney, D.G. Weber, Randall Brown,
Odyssey Research Associates
301 A Dates Drive; Ithaca, NY 14850
and
Rammohan Varadarajan
Informix Software, Inc.
4100 Bohannon Drive; Menlo Park, CA 94025

Abstract

Extension in the Trusted Heterogeneous Architecture (THETA) is accomplished by the introduction of new types and type managers. We outline a method to automate development of type managers in THETA. If types are supported by multi-level secure (MLS) managers then the TCB would be extended. We argue that automating the extension not only enhances functionality but provides for higher security assurance. THETA renames SDOS, a Secure Distributed Operating System.

1 Introduction

The Trusted Heterogeneous Architecture (THETA), formerly known as the Secure Distributed Operating System (SDOS), is in experimental development at Odyssey Research Associates, Inc. (ORA). The system is being designed and built to meet TCSEC B3 [12] security and assurance requirements. This is in contrast to an earlier phase of the project [6], [7] which produced a design targeted towards the TCSEC A1 criteria.

THETA is intended to support many kinds of applications, but in particular, Command and Control applications potentially needed by the Air Force. These applications motivate extensibility in several ways. First, C^2 applications span many types of computer systems and require survivability, scalability and interoperability. Second, they involve diverse aspects of the use of secure information including collection, selection, aggregation and analysis. Additionally, these applications involve monitoring and controlling physical devices that collect and use secure information.

This paper focuses primarily on our philosophy and mechanisms for extensibility in THETA. We discuss in detail a methodology that helps achieve this extension with high assurance. The system overview, architecture and the security policy will be dealt with in enough detail to build the background for the emphasis of the current topic. The reader is referred to [7], [14], [15] and [5] for a detailed exposition of the system goals, design, and security policy.

*This work was supported by the Air Force Systems Command at Rome Air Development Center under Contract No F30602-86-C-0146. The views and conclusions contained in this paper are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Air Force or the U.S. Government.

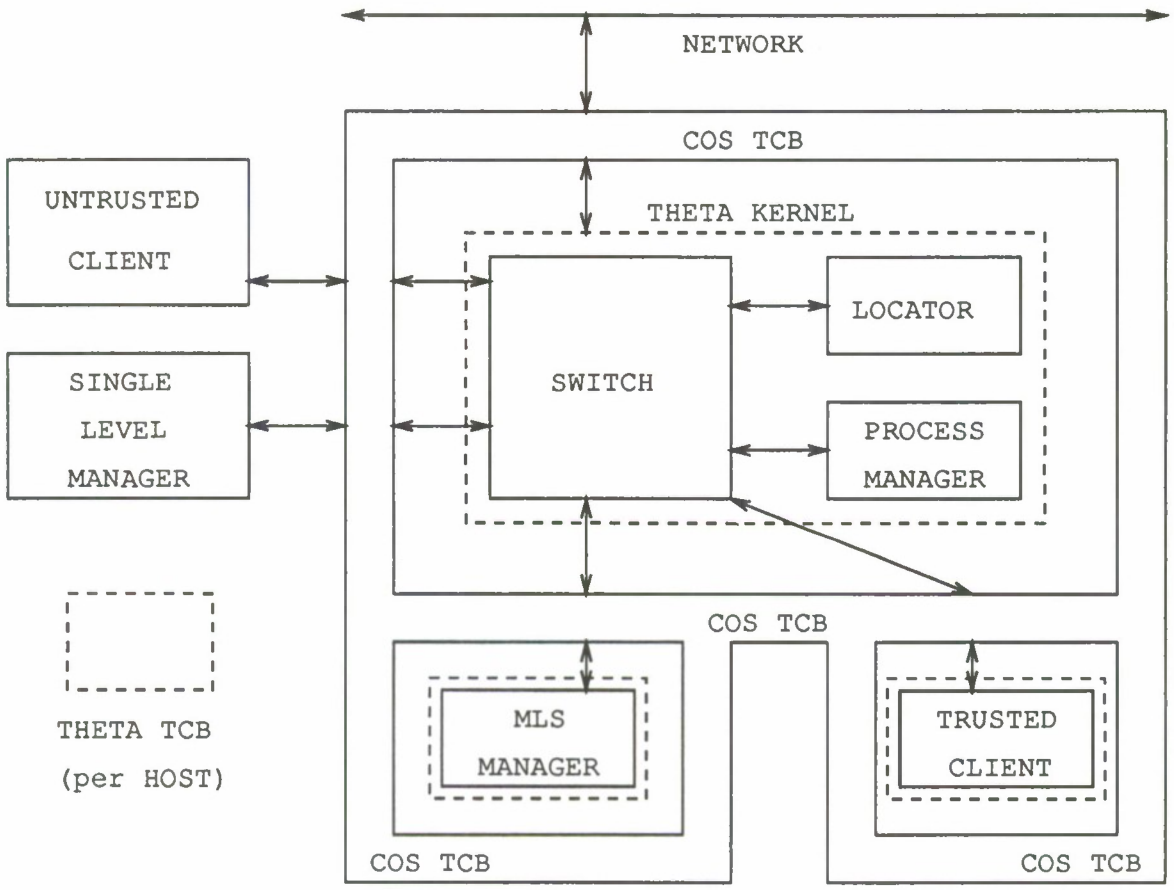


Figure 1: THETA System Components – Schematic

2 System Overview

THETA is based on the object-oriented, client-server paradigm. THETA borrows many of its concepts from Cronus, a distributed operating system developed at BBN Systems & Technologies, Inc. [2]. Indeed, the concept of auto-generation of type managers used in THETA is due to the Cronus effort at BBN. THETA, however, has been designed to provide multi-level security, enhanced subject identification, discretionary access control, configuration security, audit, COMSEC protection and TCSEC assurance.

THETA objects are instances of abstract data types. The definition of a type includes the set of operations that are possible for objects of that type. There is a hierarchy of types. Each type with the exception of the root type, has exactly one parent. A type may inherit operations from its ancestor types. A type may also define new operations.

Figure 1 illustrates the major system components and the communication paths in THETA. In this figure, the THETA TCB boundary is marked by dashed lines and only one host is shown.

Objects can be accessed by invoking operations on them. Client programs act on behalf of users to issue such invocations. THETA users interact with the system through the user interface which permits execution of THETA system client or user-written application client programs. The invocation of an operation is the only way to meaningfully access an object. Operations are implemented by type managers. A manager insulates client applications from the internal representation of objects of a given type, and provides a precisely defined interface to the object. The kernel (which is the component of THETA that runs on every THETA host) is made up of the **Switch**, **Locator**, and **Process Manager.**[1] The **Locator** is responsible for locating objects in support of location transparency offered in THETA. The **Switch** routes invocations and replies. The **Process Manager** maintains attributes of THETA processes and operations on THETA hosts. All resources in the system are represented as objects, and all operations are carried out as described above.

3 THETA Architecture

THETA is implemented using a layered architecture, which is illustrated in Figure 2. The THETA clients, managers, and the kernel processes are implemented on top of an existing trusted Constituent Operating System (COS). A COS process becomes an THETA process by interacting with the THETA kernel via the Register Process protocol (see [11]). The current design calls for THETA to be implementable without modifications to the COS. All COSs in an THETA network must meet TCSEC B3 security and assurance requirements for the combined THETA system to be B3. The following features of the COS are used:

- assured process separation — direct interprocess communication that is not controlled by the system must be disallowed. To achieve this the MAC, DAC, and user and process identification mechanisms of the COS will be used.
- non-interference with process operation — processes responsible for security must not be tampered with. The same COS mechanisms mentioned previously are used.
- stable storage — data needed for enforcing security and for maintaining object representations must be protected. The COS file system will be used to achieve this.
- IPC support — trusted path, local IPC and TCP/IP facilities of the COS are used to support THETA IPC primitives and protocols. (Note: in the initial demonstration version of the system secure transport facilities for communications networking are not available for use in the design. As an interim measure, non-secure TCP/IP was used—with the provision that the file system protections were set up so that use of TCP/IP was restricted to trusted processes *only*. In the future, trusted interhost communication at the B3 level will be needed to complete the implementation.)

4 Constituents of the TCB

The TCB for the system is the TCB's of all the COS's and the TCB that THETA introduces. The current phase of the project does not make any modifications to the COS's TCB. Since the THETA TCB is configurable, by choosing which managers are trusted, it is important to determine what is necessarily in the THETA TCB. The **Switch** is the only necessarily MLS component of THETA. The **Switch** is a small piece of software with a single thread of execution and hence does not add greatly to the size or complexity of the TCB. The configurable part, of course, involves the managers. All MLS managers will be part of the TCB. Each THETA site can determine which managers it wants to run as MLS.

[1]other transient processes are part of the kernel. We shall discuss them in a forthcoming publication of the detailed design [11].

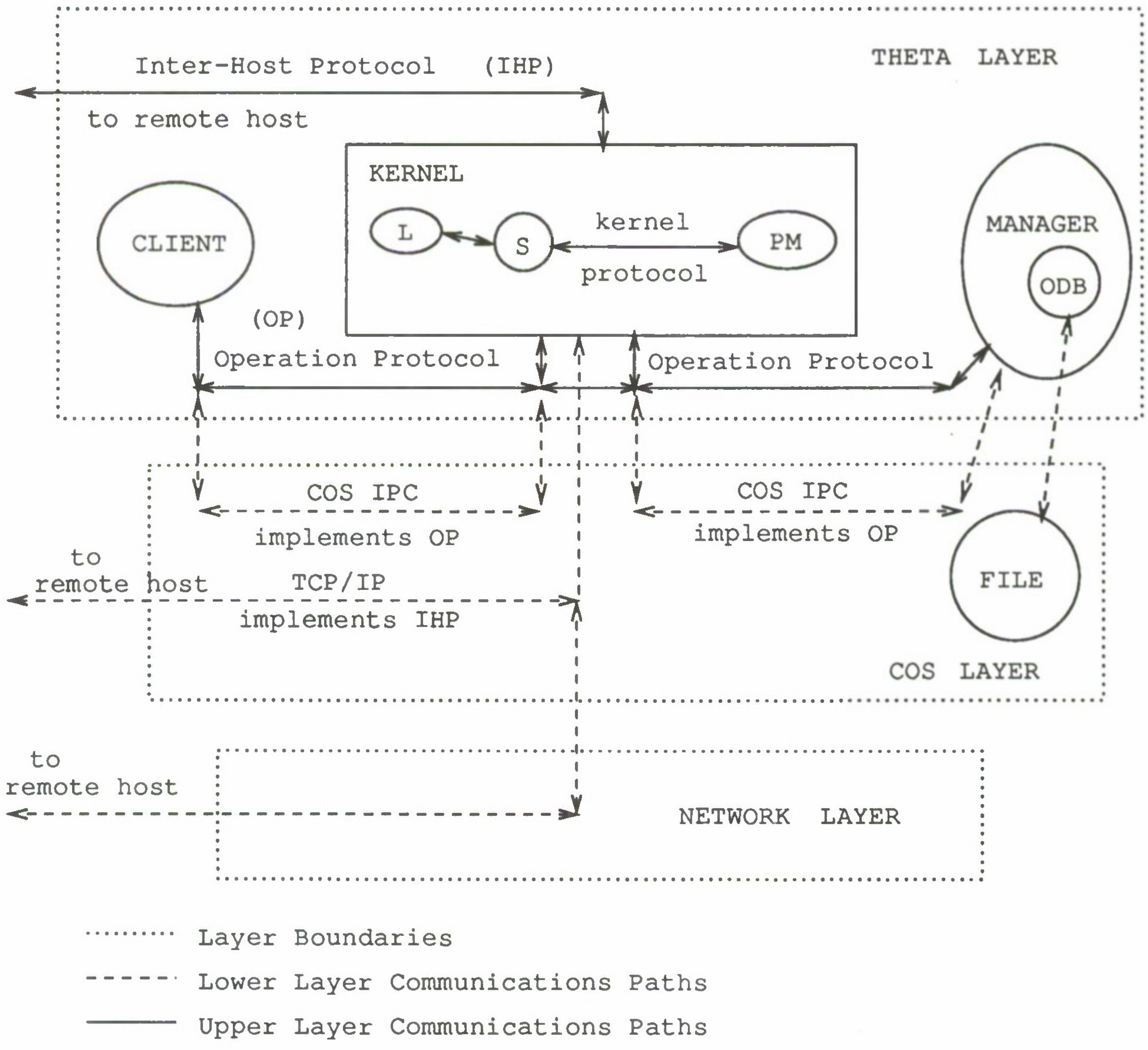


Figure 2: THETA Layering

5 Extending THETA Securely

A fielded THETA system has certain built-in types. Support for each of these types could be provided by MLS or MSL (Multiple Single Level — multi-level service offered by running a manager instance for each level in a given range). At a particular site the Security Administrator (SECADM) must decide the mix of MLS and MSL manager instances. These decisions affect the size of a host's TCB. In addition, the SECADM may decide to add to the built-in set of system types to meet particular needs. Also individual users may, with SECADM approval and manual installation assistance, create their own types and managers. As new types are introduced and managers and clients are built, the THETA system is extended. It is important that extensibility be a simple exercise that does not invalidate the trust already placed in the TCB being extended.

5.1 THETA Security Policy

The THETA security policy is outlined in [5] and formally addressed in [10]. This policy includes provisions for Discretionary Access Control and Security Administration functions as well as Mandatory Access Control. It is the MAC policy that we will consider here.

Ideally, the mandatory policy constraint on information flow is that the THETA system be restrictive [3]. Restriction is a formally defined security policy that prevents highly classified information from flowing to lower security levels, either accidently or maliciously and either through overt or covert channels [2]. Restriction is a composable property, which means that the hook-up [4] of restrictive processes within the TCB forms a larger restrictive process. Hence, so to show that the TCB is restrictive it would then be sufficient to show that every component process of the TCB is restrictive. Processes outside the TCB are at a single-level and therefore are trivially restrictive. Thus, to show that the THETA is restrictive, it is sufficient, by the hook-up property, to show that all THETA processes are restrictive. [3]

The problem is guaranteeing that the new components are restrictive. For that matter, every MLS piece of the system has to be proven restrictive. Let us examine the THETA system to identify such pieces. THETA client processes are single level entities and therefore trivially restrictive. In the kernel, the **Switch** is the only non-manager component and it does need to be MLS (restrictive). The **Switch** is a small piece of software that implements a simple design and hence can be shown restrictive without much difficulty. Single level managers (managers implemented under the MSL scheme are single level too) are trivially restrictive. That leaves the case of MLS managers. Therefore, extending THETA by adding MLS managers would entail establishing that any such managers are restrictive; the restrictivness of the extended TCB is then automatic because of composability.

Since managers can be fairly complex pieces of software, it is legitimate to ask why should they be part of the TCB? This question has been considered in [14]. The main point in favor of the MLS scheme is an increased efficiency obtained by minimizing the number of processes contending for system resources. The MSL scheme can potentially flood the host with processes for each level **AND** each type. When considered in conjunction with the IPC processes used to ensure secure communications, it is easy to see that throughput could suffer drastically. However, on those systems where the SECADM deems security issues to supercede considerations of efficiency or until MLS managers have found their place in the sun, the MSL scheme is an option that THETA will provide.

[2]probablistic information flow is not addressed.

[3]While this approach is sufficient, there are some problems in implementing it. Our approach to deal with the problems is outlined in [10]

5.2 Assurance

The astute reader may have figured out that a large chunk of manager activity would be invariant over types. Indeed a THETA manager consists primarily of

- A framework or skeleton, which consists of the process' main program, initialization functions, IPC functions, an operation processing package, audit functions, and possibly replication protocols.
- Autogenerated as well as hand coded functions that manipulate the managed objects, deal with issues of message construction and formatting, and provide a uniform user interface for the operations.

However, not all managers need detailed functionality — so they could have uncomplicated designs. If manager generation were largely automated, then a significant amount of the design and implementation is invariant over types and so can be reused. Type-specific components that provide standard functionality can be auto-generated. The security and audit checks required for specific manager operations could also be auto-generated or included in the manager skeleton and possibly both options can be employed. (We will elaborate on these in the following sections.) The assurance of security for MLS managers is now divided between the manager generation tool, which is a one-time assurance effort, and the manager operations, whose assurance must be determined on a manager by manager basis.

To explore automated extensibility further, one has to understand the design, functionality, and the implementation strategy for the managers. We shall do that in the following sections. In this discussion we shall present the main components in detail enough to help make the case for automated extensibility. The reader is referred to [11] for a detailed exposition on the managers.

6 Manager Design

We shall discuss the manager design by outlining two phases of manager operation: the initialization phase and the operational phase. Figure 3 shows the components and the interaction among them during the initialization phase. The operational phase set up is shown in Figure 4. The components that are shown in the two figures are for managers with maximum THETA functionality. (See section 6.3 for a discussion of core and optional manager functionality.)

A brief discussion of the components follows.

- Initializer: The initializer is responsible for setting the stage for the manager to manage objects. This involves creating databases for the types and security levels that the manager is responsible for, registering the manager process with the kernel, creating and starting up the network server, message server and the automatic replication tasks.
- Object Database: All THETA objects reside in the Object Database (ODB). Also present is a collection of routines by which managers access the object database. The ODB may be implemented by persistent COS files or in memory.
- Replication Protocol: The replication protocol provides for meaningful communication with the managers on the other hosts in order to maintain replicated objects in synchrony.
- Message Server: The message server is responsible for routing all messages between the kernel and the various tasks in a manager process. It also keeps track of tasks awaiting replies, starts up new tasks to service incoming invocations and audits operation invocations. In addition, since the Message Server is the main communications port between operations and the kernel, it also makes sure that the levels stamped on a message are appropriate for the intended operation.

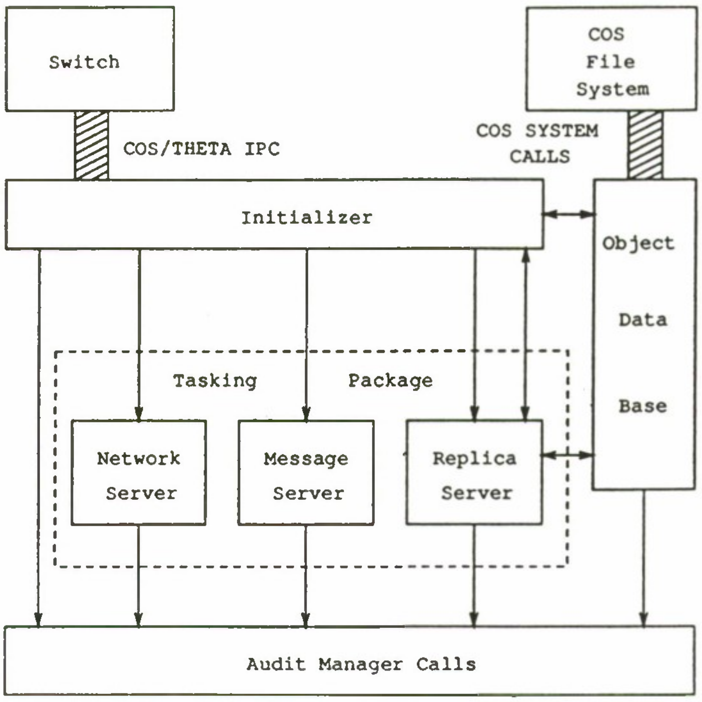


Figure 3: Manager Initialization Phase

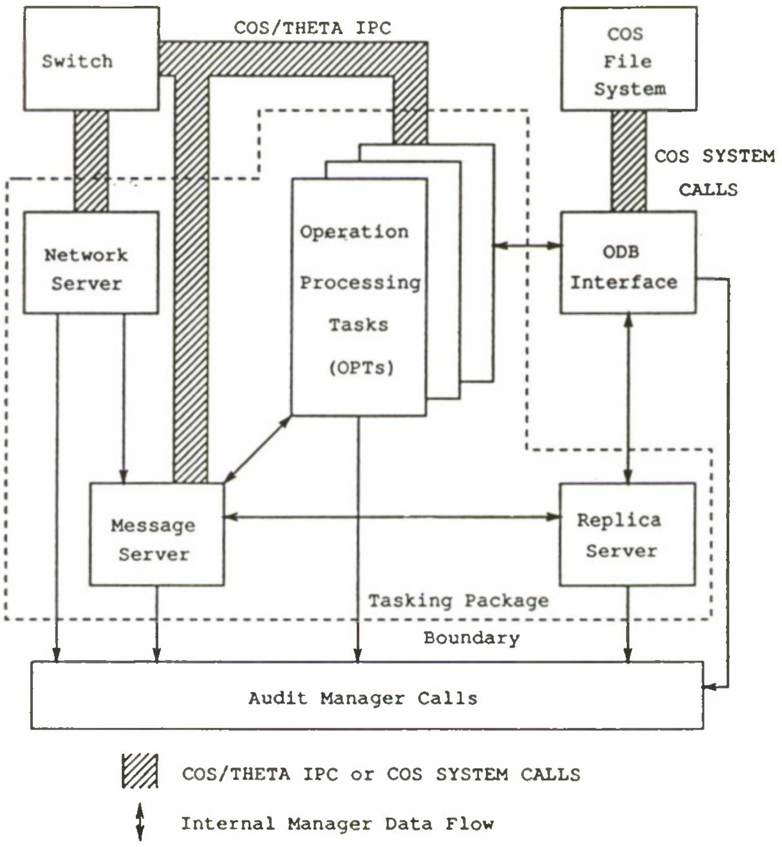


Figure 4: Manager Operational Phase

- Network Server: The network server detects IPC activity over the communication channel(s) connecting the manager to the kernel.
- Operation Processing Task: When a manager receives an operation invocation, the Message Server starts a new Operation Processing Task (OPT) to handle the request. The OPTs call the code for type dependent operations on objects, perform Mandatory Access Control (MAC) checks to ensure that the particular access is within the constraints imposed by the THETA security policy, and perform Discretionary Access Control (DAC) checks to verify that the invoker is authorized to perform the particular operation on the particular object as per the THETA DAC policy. The OPTs also perform auditing as required. It should be noted that all OPTs are constructed from a non-autogenerated task framework, called **InvokeRequest** and the hand written operation specific code. The **InvokeRequest** function is part of a manager's skeleton and will be shared by all manager operations. The manager skeleton source need not be available to manager developers, and so it will not be easy to circumvent manager MAC and audit functions. In MLS managers one must also trust the operation-specific code. However, once the skeleton code has acheived trusted status, one need only maintain its integrity.

6.3 Manager Functionality

Functionality common to all managers includes sending and receiving messages, processing messages, replication support, consistency/availability support and ODB support. Optional functionality would include security — being MLS, concurrency, and multi-tasking support. Even in the common functionality, there is a lot of freedom to tailor the manager. For instance, there are several kinds of replication support to choose from. The ODB for instance could be on disk or in memory. The THETA design approach is to: incorporate common and optional functionality as part of the trusted support library. Functionality is to be supported in a modular fashion so that users can tailor the managers with only the desired functionality, and of course — to automate development.

6.4 Manager Implementation

We have identified sizable chunks of the manager that can be selected from pre-built components or chunks that can be auto-generated. We will use a specification language in which to state the required parameters and hints. We will then build a tool that would parse the specification and build most of the files that go into making a manager.

We have collected reusable components and routines into manager support libraries. These THETA Managers support libraries have been stripped of extraneous functions to comform to the TCB minimization criterion that will be in force in the case of MLS managers. However, MSL managers will also share in this minimization, since the same manager skeleton is used in this case. These libraries include routines for Hash and Cache table management, THETA IPC, Message Formatting, and Queue Management.

As a final implementation issue, we must note that it is the operation-specific code would not be auto-generated. This will have to be hand coded.

6.5 Security Critical Issues

The address space that a manager executes in is not partitioned by security level. If managers are single level (or implemented by the MSL scheme), then the single address space poses no concern. For MLS managers, however, care must be exercised in design and implementation so as to avoid any illicit information flow.

Our approach is part brute force and part sophisticated. In section 6.4 we said that the support library is a pool of reusable components from which all managers can draw. The brute force part of ensuring security is to guarantee that the algorithms used in the reusable manager components are trusted; the security level of various data structures in a manager must be identified and the implementation of the reusable manager components must be capable of forming a restrictive manager in the presence of operations that transfer data from structures at one level to structures at greater or equal levels.

It is true that ensuring the trusted behavior of the library components is a formidable task. But the exercise has to be undertaken just once. Also note that THETA does not force use of MLS managers. If a site administration does not want to go through the assurance exercise, it is free to offer multi-level services in a MSL fashion.

Ensuring that the THETA libraries are trusted is only half the problem. Sophistication in addressing manager security comes into play in dealing with pieces other than the support library. As stated earlier (section 6.4), these components are mostly auto-generated from a specification. The operation processing routines are partly hand coded. The challenge is to assure that these are trusted. More complicated operations that access data at many levels can be useful; but assuring that these are restrictive is also more difficult.

MAC, DAC and Audit requirements are specific to every operation routine. If the manager generation tool inserted these MAC, DAC and Audit checks from hints in the specification, we could make the case for increased assurance. We would of course have to deal with security of the tool — which again is a one time exercise using brute force techniques. Additionally, if the support libraries are shown to be trustworthy, good software engineering practice of using the standard library primitives to compose the operation processing routines would contribute to high assurance of security.

7 Concluding Remarks

Trusted extensibility is natural in a kernelized, trusted system like THETA. The trusted kernel will provide all the secure functionality needed and in minimal form. However, MLS object managers are nonetheless very desirable to provide additional trusted functionality, and to increase the overall efficiency of the system. Conformance of such MLS managers to the THETA security policy (restriction) provides the formal justification that such trusted extensions preserve security. The hard problem that remains is justifying that each trusted, MLS object manager added to the system is restrictive. This is the problem addressed by this paper.

A software tool is used to generate the framework of each THETA manager automatically. Input to the tool is a specification of the operations that the manager will implement, and specifications of some properties of each operation. The specification language is not expressive enough to describe the semantics of each operation in detail, so functionality that is specific to the manager must be coded by hand and called at the appropriate points from the automatically generated code.

The goal in THETA has been to include security-relevant features of operations as part of the manager specification language. The features that can be specified include the direction of data flow (read, write, read-write), and the manager's approach to concurrency control of invocations at different security levels. The former are used to select the Bell-LaPadula access control checks automatically, and the latter are used to resolve automatically multi-level contention for resources in ways that limit or close all covert channels. These two features of the specification language are sufficient for automatically selecting the security-relevant manager code in many cases. All that remains in these cases is to show that the manager-specific code inserted for each operation does not interfere with or subvert the MAC security checks that are automatically generated.

Are the generated managers *guaranteed* to be trusted TCB extensions? No. It is still necessary to inspect the manager-specific code inserted manually for each operation. Because the specification language does not completely define the semantics of each operation, it is possible for the programmer to write code that maliciously or unintentionally changes the manager's security properties. It may be possible in the future to automate checks that reduce or (in some cases) eliminate this possibility.

References

[1] Schantz, R., Thomas, R., and Bono, G. *The Architecture of the Cronus Distributed Operating System.* Proceedings of the IEEE 6th International Conference on Distributed Computing Systems, May 1986.

[2] *Cronus: Revised System/Subsystem Specification.* BBN Report No. 5884, Revision 1.6, August 1988.

[3] McCullough, D. *Ulysses Security Properties Modeling Environment: The Theory of Security.* Odyssey Research Associates, July, 1988.

[4] McCullough, D. *Specifications for Multi-Level Security and Hook-Up Property.* Proceedings of the 1987 IEEE Symposium on Security and Privacy, April 1987, pp. 161-166.

[5] Proctor, N., and Wong, R. *The Security Policy of the SDOS Prototype.* To appear in Proc. 5th Aerospace Computer Security Conference, December 1989.

[6] BBN and ORA Staff, *The Secure Distributed Operating System Design Project.* RADC-TR-88-127, Jan. 1988.

[7] Casey, T., et al., *A Secure Distributed Operating System.* Proceeding of the 1988 IEEE Symposium on Security and Privacy, April 1988, pp. 27-38.

[8] ORA Staff, *System Segment Specification for the SDOS.* ORA Technical Report, TR 25-1. Feb. 1989.

[9] ORA Staff, *System Segment Design Document for the SDOS.* ORA Technical Report, TR 25-2. June 1989.

[10] ORA Staff, *Formal Security Model Document for the SDOS.* ORA Technical Report, TR 25-4. October 1989.

[11] ORA Staff, *Software Design Document for the SDOS.* ORA Technical Report, TR 25-5. December 1989.

[12] *DoD-5200.28-STD, DoD Trusted Computer System Evaluation Criteria.* December 1985.

[13] Weber, D.G., and Lubarsky, R., *The SDOS Project — Verifying Hook-up Security.* In Proc. 3rd Aerospace Computer Security Conference, December 1987.

[14] Wong Ray, et al., *The SDOS System Prototype*, In the 12th National Computer Security Conference, October 1989.

[15] Varadarajan, R, et al., *SDOS — An Overview*, In the 1989 Mission Critical Operating Systems Workshop, Sept. 1989.

CONTROLLING SECURITY OVERRIDES

Lee Badger
Trusted Information Systems, Inc.
3060 Washington Road (Rt. 97)
Glenwood, MD 21783

Abstract

For some critical applications, it is sometimes necessary to override security protections. Security override is in general only necessary when assets are threatened in such a direct way that security concerns are of secondary importance. In these situations, a system which does not provide a security override fails to adequately address system requirements. *Relaxation security* is a security property expressed in terms of the *guarantees* that a trusted system may provide; guarantees are statements about the conditions under which information may flow. Relaxation secure systems permit dynamic, incremental relaxation (and partial reimposition) of security constraints by authorised users. The use of guarantees permits security damage sustained during a period of constraint relaxation to be expressed in terms of guarantees violated; the set of violated guarantees may then be used as input for security recovery. This paper extends the definition of relaxation security to include relaxation of integrity policies and relaxation of supporting security requirements such as user authentication and auditing. The extended definition of relaxation security is presented using a state machine formulation. An example application demonstrates the utility of the approach. [1]

Introduction

For some critical applications, it is sometimes necessary to override security protections. Security override is in general only necessary when security controls prevent critical tasks from being performed and when assets are threatened in such a direct way that security concerns are of secondary importance. In these situations, a system which does not provide a security override fails to adequately address system requirements. Controlling such security overrides is problematic: the conditions under which security override is a lesser evil may be surprising when they occur; the inability to predict specific needs for security override precludes deciding in advance how to trade off security and other goals. An alternate approach is to allow designated users to selectively override security protections at the time when those protections conflict with more pressing system requirements. Such security overrides should be as tightly constrained as possible. In particular, it is important to reimpose security controls at the nearest opportunity to minimise security damage. For systems that provide multiple security policies, such as secrecy, integrity, and supporting policies such as user authentication and audit, it is important be able to condition the relaxation of one policy on the maintenance of another. Additionally, the security interface used to adjust security controls must be simple: users should not need to make detailed examinations of system security policy during periods in which security relaxation is necessary.

When security controls are relaxed, the security properties of the ensuing state must be examined. These fall into two broad classes: 1) measures of how much security damage has occurred, and 2) techniques for ameliorating security damage to support continuing operations. *Relaxation security* [2] is a new method of specifying security properties, which permits dynamic security relaxation for secrecy. Security specifications are expressed as sets of guarantees that a trusted system provides to its users. This paper extends relaxation

[1] This research was supported by the Defense Advanced Research Projects Agency, contract F30602-89-C-0125.

security to permit dynamic relaxation of integrity andother supporting policies. The definition of ***relaxation security*** is reviewed, and its application to integrity and other supporting policies is examined. An example demonstrates the utility of the approach. Trusted system services for supporting relaxation security are also discussed.

Related Work

Current definitions of security, e.g. [3, 9, 17, 20, 5, 1, 6, 8, 13], generally characterise security (secrecy or integrity) as a predicate that is either satisfied or not satisfied by a given system execution. Boolean valued predicates do not provide a way to specify partially secure executions. As a practical matter, trusted systems often require the ability to selectively violate abstract definitions of security through the use of trusted subjects [3], and trusted system implementations have not solved the containment problem [12]. Trusted subjects have been studied in their relation to the above definitions [14]. Special security policies, justified by the special functions performed by trusted subjects, are carried out by the (carefully studied) actions of those subjects. Covert channel analysis [12, 11, 18] addresses partial satisfaction of security definitions (for secrecy) in implementations using the metric of bits-per-second. Neither case, however, addresses dynamic, deliberate, system-wide relaxation and reimposition of security properties.

The relaxation lattices defined in [10] show how to constrain the languages accepted by automata. In [2], relaxation lattices are adapted to include a notion of information flow for secrecy but does not address relaxation of other security policies.

Security Relaxation

We model security by the set of *guarantees* that a trusted system provides to its users. Guarantees are the "promises" that a system provides to its users: guarantees are statements about the conditions under which information is (or has been) permitted to flow in a system. Secrecy and integrity policies may be specified using guarantees. In addition, guarantees may be used to assert that a system provides particular supporting policies, such as audit and authentication. The most secure system state, in this formulation, is that in which every login has passed the most rigorous authentication test, audit has been continuously enabled, and secrecy and integrity access control rules have been followed without exception. In the most secure system state, a system is able to provide a particular set of guarantees. For secrecy and (label based) integrity access controls, the guarantees specify that information has been or will be allowed to flow when certain label relationships (e.g., dominance) hold. For supporting policies, the guarantees assert that information has been or will be allowed to flow only on behalf of users that have been authenticated to a given strength (e.g., challenge-response, password, etc.), or that information flows only when the audit subsystem is functioning. After a security override, a system is able to provide a smaller set of guarantees.

Particular guarantees may be more important that others. For example, relaxation of authentication or audit may be user and (secrecy or integrity) category sensitive to prevent compromise or destruction of critical data. In addition, it may be desirable to permit relaxation of one policy (e.g., integrity) or another (e.g., user authentication), but not both. For example, a system integrity policy may be relaxed to allow emergency updates to system databases, but not by individuals who have not been strongly authenticated to the system.

We model the use of security overrides using relaxation lattices. Relaxation lattices have been used before [10] as a way to define the larger languages of operations accepted by abstract data types when security constraints have been relaxed. In [2] relaxation lattices are adapted to providing graceful security degradation for secrecy in terms of the guarantees that a system provides. Here we adapt the notion of a relaxation lattice to the problems of providing graceful security degradation for integrity, audit, and authentication policies.

Guarantees

For our purposes, a trusted system is an unbounded set of subjects S and an unbounded set of objects O with two fundamental interactions, read and write, defined for subjects and objects which comprise the only means by which information flows in the system. [2] Let L be a set of security levels on which a partial ordering, $\leq$ (and $>$), is defined. We now define several attributes for subjects and objects. Let $level : \{S \cup O\} \rightarrow L$ give the security level of a subject or an object. Let $i\_level : \{S \cup O\} \rightarrow L$ denote the integrity level of a subject or an object. Let $auth : S \rightarrow INTEGERS$ denote the strength of the user authentication associated with a subject. Finally, let *audit* be a predicate that denotes that the audit subsystem is enabled. We assume label tranquility, and model the executions of a trusted system using an automaton defined by the 4-tuple:

$$\langle STATE, s_o, OP, \delta \rangle$$

where $STATE$ is a set of (uninterpreted) states, s_o is an initial state, OP is a set of operations, and $\delta \subseteq STATE \times OP \times STATE$ is a state transition relation. $OP = \{(s,r,o),(s,w,o)\}$ where $s \in S$, $o \in O$, and (s,r,o) represents the operation in which s reads o, and (s,w,o) represents the operation in which s writes o. A system history $\alpha = \langle s_o, \pi_0, s_1, \pi_1, ..., s_n \rangle$ is an alternating sequence of states and operations such that s_o is the initial state, the s_i are in $STATE$ and each π_i is an operation in OP. In order to express cleanly how security can be relaxed, we define the *de facto* information flows for a system history. The treatment of information flow is similar to that of [4], but is developed separately here to facilitate inclusion with relaxation lattices. For $e_i, e_j \in \{S \cup O\}$, denote the flow of information from entity e_i to entity e_j in a system history α by $e_i \rightarrow_\alpha e_j$.

We define $e_i \rightarrow_\alpha e_j$ as follows:

$$e_i \rightarrow_\alpha e_j \iff \left(\begin{array}{ll} (e_i, w, e_j) \in \alpha & \vee \\ (e_j, r, e_i) \in \alpha & \vee \\ (\alpha = \alpha_1 \cdot (e_k, w, e_j) \cdot \alpha_2 \wedge e_i \rightarrow_{\alpha_1} e_k) & \vee \\ (\alpha = \alpha_1 \cdot (e_j, r, e_k) \cdot \alpha_2 \wedge e_i \rightarrow_{\alpha_1} e_k) & \end{array} \right)$$

where "$\cdot$" denotes concatenation and $\pi_i \in \alpha$ means that operation π occurs in history α.

A single state transition may induce many information flows. Let $\alpha = \alpha' \cdot \langle \pi, s \rangle$. We define the set of information flows, which may or may not already exist in α', induced by π as follows:

$$induce(\alpha, \pi) = \begin{array}{ll} \{o \rightarrow_\alpha s\} \cup & \\ \{\forall_{e:e \rightarrow_{\alpha'} o} (e \rightarrow_\alpha s)\} & if\ \pi = (s,r,o) \\ \{s \rightarrow_\alpha o\} \cup & \\ \{\forall_{e:e \rightarrow_{\alpha'} s} (e \rightarrow_\alpha o)\} & if\ \pi = (s,w,o) \end{array}$$

induce defines the set of information flows exercised by each operation. Let α^i denote the prefix of α which ends with the state following the i^{th} operation in α. The complete set of information flows which exist in a history $\alpha = \langle s_o, \pi_0, s_1, \pi_1, ..., s_n \rangle$ can be expressed in terms of *induce*:

$$\bigcup_{\pi_i \in \alpha} induce(\alpha^i, \pi_i)$$

A guarantee is a statement of the form:

$$e_i \not\rightarrow e_j$$

which is an assertion that information does not flow from e_i to e_j. A guarantee $e_i \not\rightarrow e_j$ will be satisfied by a system history α if and only if $\neg\, e_i \rightarrow_\alpha e_j$. Because we will specify security constraints which can be relaxed, guarantees will be made conditional on the absence of future user directions to relax security. A system's security policy may be specified by a set of guarantees.

Sets of guarantees may be specified using the traditional dominance relation between security levels. For example:

$$\forall_{s \in S, o \in O} \left(\begin{array}{ll} (level(s) < level(o) \Longrightarrow o \not\rightarrow s) & \wedge \\ (level(s) > level(o) \Longrightarrow s \not\rightarrow o) & \end{array} \right)$$

[2] For simplicity in the model we do not consider failed read or write attempts. For static access control rules, failed attempts transfer no information. Later, when access control rules become dynamic, we will consider failed attempts informally.

is a statement of the flow policy enforced by the ss-property and the *-property of the Bell and Lapadula model [3]. The flow policy can be expressed more succinctly:

$$\forall_{e_i,e_j}\,(level(e_i) > level(e_j) \Longrightarrow e_i \not\rightarrow e_j)$$

Guarantees may also specify integrity policies. For example, the flow policy enforced by the strict Biba [5] policy may be expresses as:

$$\forall_{e_i,e_j}\,(i\_level(e_i) < i\_level(e_j) \Longrightarrow e_i \not\rightarrow e_j)$$

Supporting policies may also be expressed with sets of guarantees. The requirement for a particular strength of authentication may be expressed as:

$$\forall_{s,e}\,(auth(s) < N \Longrightarrow s \not\rightarrow e \wedge e \not\rightarrow s)$$

This set of guarantees asserts that information will not flow to or from subjects that have not been strongly authenticated. This restriction may be modified to permit weakly authenticated subjects to observe but not modify:

$$\forall_{s,e}\,(auth(s) < N \Longrightarrow s \not\rightarrow e)$$

The requirement that information only flows when a system's audit subsystem is enabled is expressed as:

$$\forall_{e_i,e_j}\,(\neg audit \;\Longrightarrow e_i \not\rightarrow e_j)$$

A combined statement for secrecy, integrity, authentication, and audit may be given as follows:

$$\forall_{e_i,e_j}\left(\begin{array}{ll} (level(e_i) > level(e_j)) & \vee \\ (i\_level(e_i) < i\_level(e_j)) & \vee \\ (e_i \in S \wedge auth(e_i) < N) & \vee \\ (e_j \in S \wedge auth(e_j) < N) & \vee \\ (\neg audit) & \end{array}\right) \Longrightarrow e_i \not\rightarrow e_j$$

For notational simplicity, let DP denote the antecedent above. The set of guarantees that a system provides may be reduced by strengthening the antecedent as follows:

$$\forall_{e_i,e_j}\left(\left(\begin{array}{ll} DP \wedge \neg(exception_1) & \vee \\ DP \wedge \neg(exception_2) & \vee \\ \ldots & \vee \\ DP \wedge \neg(exception_n) & \end{array}\right) \Longrightarrow e_i \not\rightarrow e_j\right)$$

For example,

$$\forall_{e_i,e_j}\,(\;DP \wedge \neg(level(e_i) = Secret \wedge level(e_j) = Confidential) \qquad \Longrightarrow e_i \not\rightarrow e_j)$$

states that the desired policy holds except that information is allowed to flow down in the security lattice only when it is flowing between the classifications Secret and Confidential. Because the antecedent is stronger, fewer guarantees are specified. It is possible to relate relaxations of secrecy, integrity, authentication, and audit. In the form used above, exceptions are exclusive. They may be combined, however:

$$\forall_{e_i,e_j}\,(\;DP \wedge \neg(exception_i \wedge exception_j) \qquad \Longrightarrow e_i \not\rightarrow e_j)$$

The largest set of guarantees represents the most constrained system executions. Smaller and smaller sets of guarantees correspond to more and more relaxed security policies.

Sets of guarantees may be satisfied using a system model where state transitions only occur when their enabling conditions hold as follows: in the absence of user commands to relax security constraints, a system which provides a guarantee

$$P \Longrightarrow e_i \not\rightarrow e_j$$

must include $\neg P$ in the enabling condition of every state transition which might cause $e_i \rightarrow_\alpha e_j$ for a system history α.

It is important to note that, unless a specification places restrictions, using guarantees, on the flow of information within a single security level, the disclosure of one object at a given security level may imply the disclosure of another object at the same level because one object may be encoded in another within a security level. Similarly for label based integrity policies, the exposure of a high integrity subject to a low integrity object may imply exposure to other low integrity objects. The worst may not be realised, however. For relaxation security, the goal is to limit the possibility of such flows as much as possible, and to make available the evidence of any such flows when a security recovery is attempted.

Relaxation Lattices

A relaxation lattice, as defined in [10], is a lattice $\mathbf{A}$ of automata which are identical in every way except possibly for their state transition relations. The automata are parameterised by elements of 2^C where C is a set of security constraints which are defined as the complement of the access rights that subjects have to objects. For our purposes, C is a set of guarantees. The lattice is oriented such that the automaton which satisfies the largest set of constraints, and thus accepts the smallest language, is at the top. Each automaton A is a state machine $\langle STATE, s_o, OP, \delta\rangle$ defined as above. The "environment", which determines which set of constraints must be satisfied, is modeled by an automaton $\langle 2^C, c_o, EVENT, \delta_B\rangle$ where elements of $EVENT$ are operations which change the current set of constraints and $\delta_B \subseteq 2^C \times EVENT \times 2^C$ is a state transition relation. Let $\phi : 2^C \rightarrow \mathbf{A}$ be a lattice homomorphism. For the purposes of intentional security relaxation, we will modify this scheme slightly so that the security restrictions enforced by automaton $\phi(C_i)$, $C_i \in 2^C$, may be a subset of C_i. For intentional security relaxation, $EVENT$ is the set of special user commands which explicitly change system security constraints. The special-command automaton and lattice together are modeled by a composite automaton

$$\langle 2^C \times STATE, (c_o, s_o), EVENT \cup OP, \hat{\delta}\rangle$$

where $\hat{\delta}$ contains state transitions both to change the current set of constraints and to model accesses by subjects to objects. Let δ_A denote the state transition relation of automaton A in the lattice $\mathbf{A}$ of automata. As defined in [10], $\hat{\delta} : 2^C \times STATE \times \{EVENT \cup OP\} \rightarrow 2^C \times 2^{STATE}$ is defined by two component state transition relations $\delta_1 : 2^C \times \{EVENT \cup OP\} \rightarrow 2^C$ and $\delta_2 : 2^C \times STATE \times \{EVENT \cup OP\} \rightarrow 2^{STATE}$ such that:

$$\begin{aligned} \delta_1(c,p) = &\quad \textit{if } p \in EVENT \textit{ then } \delta_B(c,p) \textit{ else } c \\ \delta_2(c,s,p) = &\quad \textit{if } p \in OP \ \wedge \ A = \phi(\delta_1(c,p)) \textit{ then } \delta_A(s,p) \\ &\quad \textit{else } \{s\} \end{aligned}$$

where $\delta_B(c_1,p)$ denotes a c_2 such that $(c_1,p,c_2) \in \delta_B$ and $\delta_A(s_1,op)$ denotes an s_2 such that $(s_1,op,s_2) \in \delta_A$. Note that, if an operation is in both OP and $EVENT$, the constraint state is changed first, and the δ relation for the appropriate automaton is then selected for the new environment state.

Let $\alpha = \alpha_1 \cdot \alpha_2$ such that the first operation of α_2 is the last operation of α which is in $EVENT$. A system which provides a guarantee $P \Longrightarrow e_i \not\rightarrow e_j$ during α_2 must include $\neg P$ in the enabling condition of every state transition in α_2 which might cause $e_i \rightarrow_\alpha e_j$. It is not necessary for $\neg P$ to have been in the enabling conditions of state transitions during executions which are prefixes of α_1: it is only necessary that state transitions that would have induced the flow did not in fact occur.

Relaxation Security

As in [2], we present relaxation security using a modified relaxation lattice. The lattice is modified to incorporate constraints expressed in terms of past system history. Elements of 2^C will not totally define security constraints, but will instead serve more as statements of user intent which are satisfied to the extent possible by the system.

A trusted system which supports security relaxation may provide different sets of guarantees at different times. Let $G_1 \subset G_2$ be two sets of guarantees. A trusted system which initially provides the guarantees in G_2 and then only the guarantees in G_1 may or may not be able to return to providing the guarantees in G_2 depending on whether or not guarantees in $G_2 - G_1$ have been violated. Let $\alpha = \langle (s_o, c_o), \pi_0, (s_1, c_1), \pi_1, ..., (s_n, c_n) \rangle$ be a system history of the composite automaton. Let $C_{op}(\alpha)$ denote the index of the last operation of α which is in $EVENT$, or 0 if α contains no operations in $EVENT$. Let $C(\alpha)$ give the set of constraints established by the last operation of α which is in $EVENT$, or c_o if α contains no operations in $EVENT$. As before, denote the prefix of α which ends with the state following the i^{th} operation π_i by α^i.

We say that a system history α of length n is *relaxation secure* if:

$$\forall_{0 \leq k \leq n} \forall_{(e_i \rightarrow_{\alpha^k} e_j) \in induce(\alpha^k, \pi_k)} \left(\begin{array}{ll} (e_i \not\rightarrow e_j \notin C(\alpha^k)) & \vee \\ \exists_{\pi_l \in \alpha^k} \left(\begin{array}{ll} l < C_{op}(\alpha^k) & \wedge \\ e_i \rightarrow_{\alpha^l} e_j \in induce(\alpha^l, \pi_l) & \end{array} \right) & \end{array} \right)$$

The meaning of this definition is that, during a period in which the set of guarantees that the system should provide does not change, a relaxation secure system prohibits the violation of guarantees that the system is still able to support. Intuitively, a relaxation secure system moves through a number of phases, providing a particular set of guarantees in each phase. In each phase, a relaxation secure system will prohibit those operations that would violate currently promised guarantees *which have not already been violated* in previous, more relaxed phases. An intuitive and immediate objection to this definition of security is that a violated guarantee can apparently be exploited in all subsequent system phases. A consequence of the definition, however, is that future exploitation is confined to the still-executing subjects which incurred the original violations. In addition, exploitation is limited to causing information flows between subjects and objects which have already experienced information flows: new subjects and objects may not be included. This "grandfathering" of relaxed subjects and the objects that they manipulate permits a system to move to a less relaxed mode of operation without immediately halting the progress of subjects which are violating no additional guarantees and whose execution was deemed important enough to initially relax security. (From a worst case viewpoint, the continued activity of these subjects is not significant since all the damage occurred on the first access.)

This definition is motivated by the need to provide guarantees about what has *not* happened in a system which permits security relaxation, and also by the need for flexibility in allowing security relaxations which are directed by a simple user interface. A user should not need to make detailed examinations of a system security policy during a time when security relaxation is necessary. A user's preferred interaction is to notice that some important job cannot be performed because of the security policy, relax the security policy using simple commands, notice that the job is being performed, and then restore security guarantees to the extent possible while allowing the important job to continue. If the important job could be identified beforehand, it would be possible to design trusted subjects to perform it; the fact that such jobs may only become apparent during a crisis necessitates the ability to globally and dynamically relax security restrictions.

A consequence of this definition is that high users may choose to influence the access control decisions made in later, more constrained, phases of system execution. If a high user writes into a low object, low users that have not already read from that object will be prohibited from reading from it after the relaxation is rescinded. This information flow occurs when accesses fail (not reflected in the automaton model), and constitutes a covert storage channel. This channel can be controlled by delaying failed access attempts.

The definition of relaxation security is not appropriate for all applications. For instance, if o_j is an object which models a device connecting a trusted system to its external environment, and if $o_i \not\rightarrow o_j$ is a guarantee that can no longer be provided, the continued flow of information from o_i to o_j represents the continued export [21] of information from the system. Even though the original guarantee cannot be provided, further flow may not be desirable. To constrain such behavior, we introduce a set of "strong" constraints which must be honored regardless of past violations. This set requires a modification to the relaxation lattice, which is now a lattice $\mathbf{A}$ of automata which are parameterised by elements of $2^C \times 2^C$. As before, the automata in $\mathbf{A}$ are identical in every way except possibly for their state transition relations. We model state transitions between strong system constraints using a state machine defined by the 4-tuple $\langle 2^C, c_o, EVENT', \delta_{B'} \rangle$, and compose this state machine with the original:

$$\langle 2^C \times 2^C \times STATE, ((c_o, c_o'), s_o), EVENT \cup EVENT' \cup OP, \hat{\delta} \rangle$$

where $EVENT'$ is the set of commands that set constraints which must be satisfied by the system regardless of any past violations. Similar to the definition above, $\hat{\delta} : (2^C \times 2^C) \times STATE \times \{EVENT \cup EVENT' \cup OP\} \rightarrow (2^C \times 2^C) \times 2^{STATE}$ is defined by two component state transition relations $\delta_1 : (2^C \times 2^C) \times \{EVENT \cup EVENT' \cup OP\} \rightarrow (2^C \times 2^C)$ and $\delta_2 : (2^C \times 2^C) \times STATE \times \{EVENT \cup EVENT' \cup OP\} \rightarrow 2^{STATE}$ such that:

$$\begin{aligned}
\delta_1((c, c'), p) = \quad & if\ p \in EVENT\ then\ (\delta_B(c, p), c') \\
& else\ if\ p \in EVENT'\ then\ (c, \delta_{B'}(c', p)) \\
& else\ (c, c') \\
\delta_2((c, c'), s, p) = \quad & if\ p \in OP\ \wedge\ A = \phi'(\delta_1((c, c'), p))\ then \\
& \delta_A(s, p)\ else\ \{s\}
\end{aligned}$$

where $\phi' : (2^C \times 2^C) \rightarrow \mathbf{A}$ is a lattice homomorphism.

This lattice provides, in essence, another lever for security relaxation. Legal system histories are easily defined for the new composition. Let $\alpha = \langle ((c_o, c_o'), s_o), \pi_0, ((c_1, c_1'), s_1), \pi_1, \ldots, ((c_n, c_n'), s_n) \rangle$ be a system history. We require one additional definition: let $C'(\alpha)$ denote the set of strong constraints established by the last operation of α which is in $EVENT'$, or c_o' if no event of α is in $EVENT'$. A system history α of length n is ***strong relaxation secure*** if:

$$\forall_{0 \leq k \leq n} \forall_{(e_i \rightarrow_{\alpha^k} e_j) \in\, induce(\alpha^k, \pi_k)} \left(\begin{array}{lr} (e_i \not\rightarrow e_j \notin\ C'(\alpha^k)) & \wedge \\ \left(\begin{array}{lr} (e_i \not\rightarrow e_j \notin\ C(\alpha^k)) & \vee \\ \exists_{\pi_l \in \alpha^k} \left(\begin{array}{l} l < C_{op}(\alpha^k) \\ e_i \rightarrow_{\alpha^l} e_j \in induce(\alpha^l, \pi_l) \end{array} \quad \wedge \right) & \end{array} \right) & \end{array} \right)$$

This definition is very similar to that for relaxation security. The only difference is that the system state includes two dynamic sets of constraints. In addition to satisfying the definition for relaxation security with respect to the original constraints, a system must always satisfy the current set of strong constraints. The access control rule that would produce strong relaxation secure histories can be informally stated: deny access if (access would violate a current strong constraint) or ((access would violate a guarantee that is currently required) and (the guarantee has not been violated during past relaxations)).

Security Recovery

The ability to relax and reimpose security constraints defines a partial security recovery scenario: when constraints are reimposed, recovery occurs automatically to the extent that no security damage occurred. When $e_i \rightarrow_\alpha e_j$ for an execution α and $e_i \not\rightarrow e_j$ is in C, however, the flow $e_i \rightarrow_\alpha e_j$ represents security damage which must be accounted for. In part, the accounting is automatic: if $e_j \not\rightarrow e_k$ is not in C, but $e_i \not\rightarrow e_k$ is in C, operations which induce $e_j \rightarrow e_k$ will also induce $e_i \rightarrow e_k$ and will be prohibited. After the return to a less relaxed security policy, the transitivity of information flow constraints imposes a partial isolation policy for subjects and objects which have previously violated security. The isolation policy imposes a limit on the effects that violated guarantees may have, whether the guarantees asserted a secrecy, integrity, or other policy. This mechanism effects partial recovery at the cost of a reduction in availability.

A more active recovery is required to enable the use of information that might have been mislabeled or corrupted by an unidentified user or manipulated while auditing was disabled . If sensitive information has been exported to an inappropriate external environment, or if low integrity information has been exported to a device that expects high integrity information, recovery of the information is not possible, although the TCB may provide assistance concerning the ultimate target of the information. If possibly mislabeled information has not been exported, recovery to a set of constraints $C_i \subseteq C$ is achievable if every subject and object into which information flowed in violation of a constraint in C_i was checkpointed during system relaxation and if system integrity constraints will permit a rollback to the state of these objects before the period of constraint relaxation. In this case, recovery is accomplished by deletion of possibly mislabeled objects and substitution of the checkpointed versions. In some cases, it is likely that rollback will not be feasible, and manual review of mislabeled subjects and objects will be necessary to reestablish security.

Typically [22, 23, 16, 7], mandatory access control checking is performed when an access descriptor for an object is obtained by a subject. In order to make access control sensitive to system history, it is necessary to keep track of which information flows are induced by individual accesses. For a subject which obtains current access to an additional object and then attempts to write to an object for which access has already been obtained, a trusted computing base must ensure that the new write access does not induce any currently illegal information flows. In addition, if a subject A has a descriptor (conferring read access) to an object O and subject B writes to O, access checking must be performed at A's next read from O to ensure that no illegal flows are established between subject A and the objects accessed by B. Following A's read from O, access checking must be performed for A when A attempts to write (for the first time) objects other than O to ensure that no illegal flows are established between the subjects and objects flowing into O and the objects that A writes. The overhead for this mechanism may be greater than that for the typical mechanism in which access checking is performed only at the first access. The extra overhead seems acceptable, however, because all access checking is still "triggered" by the operations which acquire access descriptors (e.g., open()): actual reads and writes do not require additional checking.

A portion of the flow information used for access control may be recorded by a trusted computer system's audit subsystem for use during security recovery. Specifically, information flow from objects of particular interest may be examined during security recovery, without examining all system objects which may be mislabeled, to determine whether or not crucial information has been disclosed, and, if so, where and to whom.

Example Application

Consider a system in which there are three secrecy levels and two integrity levels. As in figure 1, denote entity i (subject or object) with secrecy level X and integrity level Y by $E_i^{X,Y}$. Let $auth(M_2) = N - 1$ and let $auth(e) = N$ for the other entities, and assume that audit is continuously enabled.

Three sets of guarantees are relevant, the desired set of guarantees, represented by DP above, the set of guarantees in the current constraint set (CUR), and the guarantees that the system is actually able to supply (ACT). For this example, we ignore strong constraints.

Initially, $DP = CUR = ACT$ and no access may occur that would violate a guarantee in DP. A designated user may relax the secrecy portion of the desired policy by setting CUR to:

$$\forall_{e_i,e_j} \left(\; DP \wedge \neg(level(e_i) = M \wedge level(e_j) = L) \quad \Longrightarrow e_i \not\rightarrow e_j \right)$$

At this point, information may flow from E_4 to E_7 (edge 1). If information then flows from E_7 to E_8 (edge 2), an indirect flow occurs between E_4 and E_8 (edge 3). At this juncture, ACT is:

$$\forall_{e_i,e_j} \left(\; DP \wedge \neg(e_i = E_4 \wedge e_j \in \{E_7, E_8\}) \quad \Longrightarrow e_i \not\rightarrow e_j \right)$$

which is stronger than CUR but weaker than DP. Successive commands may relax authentication controls and integrity controls as follows:

$$\forall_{e_i,e_j} \left(\begin{array}{ll} DP \wedge \neg(level(e_i) = M \wedge level(e_j) = L) & \vee \\ DP \wedge \neg(auth(e_j) = N - 1) & \vee \quad \Longrightarrow e_i \not\rightarrow e_j \\ DP \wedge \neg(i\_level(e_i) = M \wedge i\_level(e_j) = H) & \end{array} \right)$$

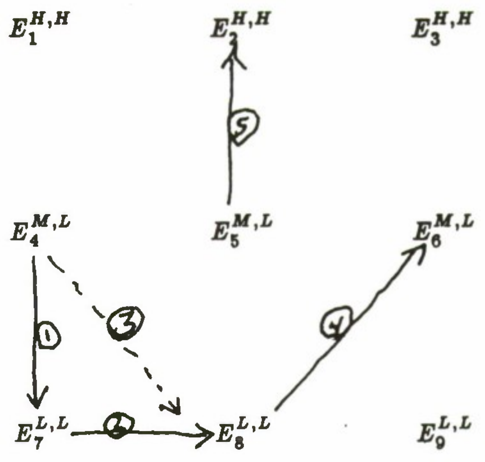


Figure 1: System Entities

At this juncture, information may flow from E_8 to the weakly authenticated E_6 (edge 4). Although authentication controls and secrecy controls have been relaxed, E_6 cannot downgrade information because the relaxations are exclusive. Information may then flow from E_4 to E_2 in violation of the desired integrity policy (edge 5). Information may not flow from E_6 to E_2, however. At this point, ACT is:

$$\forall_{e_i,e_j}\left(DP \wedge \neg \begin{pmatrix} e_i = E_4 \wedge e_j \in \{E_7, E_8\} & \vee \\ e_i \in \{E_4, E_7, E_8\} \wedge e_j = E_6 & \vee \\ e_i = E_5 \wedge e_j = E_2 & \end{pmatrix} \implies e_i \not\rightarrow e_j \right)$$

If CUR is then reset to DP, a partial isolation policy is enforced based on what relaxations occurred earlier. ACT will be unchanged: accesses that do not violate the currently promised set of guarantees will be permitted. Accesses that would cause new guarantees to be violated are prevented, however. Some accesses that would have been legal before the relaxations occurred will now not be permitted. For example, information may no longer flow from E_2 to E_1 because that would extend the information flow (contrary to the integrity policy) from E_5. Similarly, information may not flow from E_8 to E_9 because that would extend the flow from E_4 (contrary to the secrecy policy). These access prohibitions demonstrate the tradeoff between increased availability during relaxations and decreased availability afterwards. Availability may be restored through rollback, by deleting corrupted entities and restoring checkpointed versions, or (in the worst case) by manual review.

Conclusions

This paper has described the use of relaxation lattices and guarantees to specify the security properties for trusted systems during and after security overrides. Relaxation security has been defined for access controls for secrecy, label based integrity, audit, user authentication, and combinations of these policies. Transaction oriented integrity policies [1, 8] seem also amenable to graceful degradation, although their specification will require changes to the state machine formulation presented here. Future plans include further exploration of efficient algorithms to support relaxation security, and the mapping of those algorithms onto the port and task abstractions of the Trusted Mach [7] message passing architecture.

Acknowledgments

The author would like to thank Glen Benson, Martha Branstad, Jim Gray, Brian Hubbard, Tim Redmond, Dan Sterne, and Dawn Wolcott for helpful comments on the technical content and presentation.

*

References

[1] L. Badger, "A Model for Specifying Multi-Granularity Integrity Policies," Proceedings of the 1989 IEEE Symposium on Security and Privacy, Oakland, Cal., 1989.

[2] L. Badger, "Providing A Flexible Security Override For Trusted Systems,"Proceedings of the Computer Security Foundations Workshop III, Franconia, New Hampshire, P.115, 1990.

[3] D.E. Bell and L. Lapadula, "Secure Computer System: Unified Exposition and Multics Interpretation." (Technical Report No. ESD-TR-75-306, Electronics Systems Division, AFSC, Hanscom AF Base, Bedford MA, 1976).

[4] M.A. Bishop, "Practical Take-Grant Systems: Do They Exist?", Ph.D. Dissertation, Purdue University, May 1984.

[5] K.J. Biba, "Integrity Considerations for Secure Computer Systems," USAF Electronic Systems Division, Bedford, Mass., ESD-TR-76-372, 1977.

[6] W.E. Boebert and R.Y. Kain, "A Practical Alternative to Hierarchical Integrity Policies," Proceedings of the 8th National Computer Security Conference, Gaithersburg, Md., P. 18, 1985.

[7] M. Branstad, H. Tajalli, F. Mayer, and D. Dalva, "Access Mediation in a Message Passing Kernel," Proceedings of the 1989 IEEE Symposium on Security and Privacy, Oakland, Cal. P. 66, 1989.

[8] D.D Clark and D.R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," Proceedings of the 1987 IEEE Symposium on Security and Privacy, Oakland, Cal., 1987.

[9] J.A. Goguen and J. Meseguer, "Unwinding and Inference Control." Proceedings of the 1984 IEEE Symposium on Security and Privacy, 1984.

[10] M.P. Herlihy and J.M. Wing. "Specifying Security Constraints with Relaxation Lattices", Proceedings of The Computer Security Foundations Workshop II, 6-11-89.

[11] R.A. Kemmerer, "Shared Resource Matrix Methodology: A Practical Approach to Identifying Covert Channels," ACM Trans. Comput. Syst., vol. 1, p. 256-277, Aug. 1983.

[12] B.W. Lampson, "A Note on the Confinement Problem," Comm. ACM, Vol. 16, No. 10 (Oct. 1973), 613-615.

[13] S.B. Lipner, "Non-Discretionary Controls for Commercial Applications," Proceedings of the 1982 IEEE Symposium on Security and Privacy, Oakland, Cal. P. 2, 1982.

[14] J. Landauer, T. Redmond, and T. Benzel, "Formal Policies for Trusted Processes," Proceedings of the Computer Security Foundations Workshop II, Franconia, New Hampshire, P.31, 1989.

[15] C.E. Landwehr, C.L. Heitmeyer, and J. McLean, "A Security Model for Military Message Systems," ACM Transactions on Computer Systems, Vol. 2, No. 3, August 1984, pp. 198-222.

[16] G.L. Luckenbaugh, V.D. Gligor, L.J. Dotterer, C.S.Chandersekaran, N. Vasudevan, "Interpretation of the Bell and Lapadula Model for Secure Xenix," Proceedings of the 9th National Computer Security Conference, Sept. 1986, p113.

[17] D. McCullough, "Specifications for Multi-Level Security and a Hook-Up Property," Proceedings of the 1987 IEEE Symposium on Security and Privacy, 1987.

[18] J.K. Millen, "Finite-State Noiseless Covert Channels,"Proceedings of the Computer Security Foundations Workshop II, Franconia, New Hampshire, P.81, 1989.

[19] National Computer Security Center, "Department of Defense Password Management Guideline," CSC-STD-002-85. December 1985.

[20] D. Sutherland, "A Model of Information," Proceedings of the 9th National Computer Security Conference, Sept. 1986, p. 175.

[21] National Computer Security Center, "Department of Defense Trusted Computer System Evaluation Criteria," DoD 5200.28-STD, December 1985.

[22] Final Evaluation Report of Scomp Secure Communications Processor STOP Release 2.1, Sept. 23, 1985, CSC-EPL-85/001.

[23] Final Evaluation Report of Honeywell Multics MR11.0, June 1, 1986, CSC-EPL-85/003.

LATTICES, POLICIES, AND IMPLEMENTATIONS

D. Elliott Bell

Trusted Information Systems, Incorporated
3060 Washington Road
Glenwood, Maryland 21738

Abstract:

The original description of military security policies in terms of lattice theory has led to the identification of lattices both with the policy and a particular implementation technique. The position is advanced herein that diversity in lattice characterizations leads flexibility and generality to lattice-based policies and implementations. Furthermore, that set of lattice-based policies is wider than is generally recognized.

INTRODUCTION

In the field of computer security, the use of the term "lattice" to describe the nondiscretionary access control policy such as that embodied in military and intelligence policies has dated from the early 1970s (see especially [DENN76]). Unfortunately, the term itself has become synonymous in some circles with "military-access-control-policy". Where military applications or connotations are deemed inappropriate, that association has made "lattice" a code-word and red flag.

The simplest form of nondiscretionary access control policy within the military and intelligence community is expressed as a combination of hierarchical classifications (of documents) and clearances (of people) and non-hierarchical categories. Access to a physical report requires that the highest hierarchial clearance of the candidate reader be greater than or equal to the classification of the report and that every non-hierarchial category governing access to it be held by the candidate. In the simplest mathematical terms, that combination of requirements can be expressed in terms of the cross-product of two partially ordered sets, the totally ordered classification/clearance set and the set of categories, ordered by set inclusion. With the minor addition of least common dominating element and greatest common dominated element (mathematically, a least upper bound and a greatest lower bound, respectively), a lattice results.

The characterization of this simple version of nondiscretionary access control policy as a lattice policy had several benefits. One was the legitimacy conferred by pre-existing mathematical terminology. Another was the ability to represent this structure in a very efficient way within computer systems. The totally ordered component can be represented as an integer within a range, with partial-order comparisons being integer comparisons. A set of categories can be represented as a bit-map, with an ON bit representing the presence of the category assigned to that position. The bit-map mathematically was a characteristic function for the element in question. Comparison between bit-maps also an efficient analogue in computer systems, namely logical **AND**ing (or **OR**ing) of two bit-maps. Unfortunately, the relation between the lattice theory itself, the policy it was first used to describe, and the actual implementation method led to a certain degree of identification of the three. In the confusion, the useful tool of lattice theory for the description of policies and for guidance in implementations was considered more limited than it is.

The paper begins with a set of results from general lattice theory. The intent is to provide the context within which discussion of lattice policies in computer security can proceed. Next, the implications of those results on representable policies will be addressed. Finally, the theoretical and practical implications of design choices for implementing lattice policies will be described.

LATTICES [1]

A lattice can be characterized in several ways. The traditional definition of a lattice is phrased in terms of a partially ordered set L [2]. A lattice is a partially ordered set L any two of whose elements x and y have a "meet" $x \cap y$ and a "join" $x \cup y$. [BIRK48, p. 16] A useful concept in discussing lattices with this definition is that of a "cover" with respect to a partial order: an element a "covers" be provided $a \geq b$ and there is no element x such that $a > x > b$. It is immediate that if the set L is finite, then the partial order is itself characterized by the covering relation. It can be shown that the partial order $x \geq y$ is equivalent to the condition $x \cap y = y$.

The more specialized lattices of interest here are distributive lattices. A distributive lattice is one in which meet and join distribute over each other. A complemented lattice is defined as follows:

> A lattice L is complemented provided it has both an O and an I [3] and for every $x \in L$ there is $y \in L$ such that $x \cap y = O$ and $x \cup y = I$. y is called the complement to x.

A complemented lattice allows the inclusion of the concept of "not". In fact, a distributive, complemented lattice is called a Boolean lattice.

The characterization of these forms of lattice is that a finite distributive lattice is isomorphic to a ring of sets. [4] Assuming the Axiom of Choice, all distributive lattices are isomorphic to a ring of sets. [BIML65] Thus, consideration of rings of sets, special subsets of full power sets, suffices for the study of distributive lattices. Furthermore, adding the characteristic of "complemented" makes the result even stronger. Every finite Boolean algebra L is isomorphic to 2^n for some positive integer n. [BIBA70, p. 278]

Every Boolean algebra L is generated by its set of next-to-least elements, its atoms. [5] This parallels the result that a distributive lattice is generated by its meet-irreducible (dually, its join-irreducible) elements. In practical terms, the entire lattice can be generated by the elements that are "at the bottom" in the sense of not being the meet of any two other elements. In the familiar case of the power set of a finite set S, the singleton sets of S constitute a set of meet-irreducible elements.

Thus, the distinct ways that a Boolean lattice (that is, a distributive, complemented lattice) can be defined include (1) use of an explicit partial order, (2) use of explicit meet and join, defined either globally or as the transitive closure of a cover operation, or (3) **AND**, **OR**, and **NOT** operators. Any definitional base will yield the same lattice structure.

POLICIES

Any policy capable of being represented abstractly as a lattice can be termed a "lattice policy". As indicated in the section above, this usage is much broader than that usually connoted by the term. Specifically, efforts to argue the wider applicability of the "lattice access control model" (see for example [LIPN82], [LEE88]) have had to combat the identification of the particular version of lattice policy constructed for military use as well as make their own points.[6] The different characterizations of Boolean lattices, in fact, admit any policy that can be described with **AND**s, **OR**s, and **NOT**s as a "lattice" policy.

---

[1] See the Appendix for more complete definitions and some basic results.

[2] A partial order on the set L is a relation between elements that is reflexive, transitive, and antisymmetric.

[3] The element I is the lattice maximum element of the lattice and the element O is the minimum element.

[4] A ring of sets is a collection of sets closed under union and intersection.

[5] Technically, atoms are elements that cover O.

[6] As been noted frequently, the non-classified military requirements involve exactly the same concerns of isolation and separation of function as is true outside the military and outside the government.

Consider a typical example of an enterprise that formally recognizes (at least) the three information types PLANS, FINANCIAL, and OPERATIONS. From these types of information, viewed as nonhierarchical categories in the usual "lattice model" formulation, one can construct a full powerset lattice with the category $P \cup F \cup O$ as the maximum element (the I element in lattice theory) and $\varnothing$ as the minimum element (the O element in lattice theory).

It is frequently observed that this lattice model cannot represent the concept of information available to staff cleared for PLANS *or* to staff cleared for FINANCIAL. In fact, the lattice model can represent that situation, just not with P, F, and O as the meet-irreducibles. The proper meet-irreducibles are $P \cap F$, $F \cap O$, and $O \cap P$. It is the parochial view that causes this problem, identifying the basic elements of policy characterization (in this case, P, F, and O) with the lattice's meet-irreducibles. The injection in this case is to the covers of the meet-irreducibles.

In general, any policy that can be patently and easily represented in terms of partial order and meets and joins; or covers and meets and joins; or in terms of naive logic ($\wedge$, $\vee$, $\neg$) is a lattice policy and a representation in any other form, modulo the presence of required side conditions, is equivalent to a representation in any other form. This observation leads naturally to the question of what advantages accrue from different implementation approaches.

IMPLEMENTATIONS

An implementation of a lattice policy need not look exactly like any one definitional form of lattices in the abstract. Indeed, there being several different-seeming characterizations of useful classes of lattices, there are different ways of building an implementation to represent lattice policies. The choice of which implementation method to use can take into consideration both the intended customer base and the design and implementation implications themselves. The use of an implementation that is optimized for the use expected from the most desired customer segment, for example, would be of considerable advantage.

In the current field of trusted products, the implementation strategy has been largely that of representing the lattice directly as a duple of a totally-ordered hierarchical component and a bit-map representation of a set of categories. The major speed advantages in the late 1960's and the 1970's have become less important, but the implementation approach has been largely left the same. In fact, the usual explanation for the guideline figures for numbers of hierarchical classifications (8) and non-hierarchical categories (29) [TCSEC83] is presumed to be the packing label information into 4 bytes using a 3-bit integer and a 29-bit bit-map.

Other representations of security label information are beginning to be seen more regularly. Representative was the original use of the group abstraction for the provision of security levels within the AT&T UNIX. [7] [FLIN87] A second example was an implementation of three "categories" in a networking situation. In that case, the error-detection reasons, the category set NO-CATEGORIES was encoded with a fourth bit-pattern to avoid an error condition from being interpreted as NO-CATEGORIES. The representation, therefore, was not a patent and direct implementation of the bit-map view of the "lattice model" and led to a minor misunderstanding wherein the implementors had to explain that, while the bit-map was not the usual one, the underlying policy remained the expected lattice policy.

An implementor who chooses the traditional military / TCSEC duple as the paradigm for the implementation need not totally write off the customer base that prefers to think of the policy in terms of naive logic, for example. With the provision of conversion tools, to allow the logic-customers to specify and interpret the policy parameters in their own terms, the underlying base can be traditional, hiding that fact from the users. Analogously, an implementor that choose to use the logic paradigm can (if desired) provide a user interface to allow the traditional customers to manipulate and use the system with their own perspective. An implementor who chooses to implement abstract data types for security levels and a defined function **meet** (or **join**), calculating dominance as the condition $x \cap y = y$ (respectively, $x \cup y = x$) can serve both communities with proper user interface functionality at a highly isolated spot within the TCB. The implementor is not limited to one implementation approach for each market segment, but has choices in base approach as well as in the policy-conversion options to provide to make the resulting product more attractive than its competitors.

One implication of implementing systems to support lattice policies beyond the usual military classification situation

[7] UNIX is a registered trademark of AT&T.

necessity for larger lattices. Systems sized for only dozens of categories will quickly become saturated in usages such as [LEE88]. The bit-map implementation undergoes a state explosion, but other implementations, especially an abstract logic one, need not be similarly affected.

Another point sometimes raised in terms of lattice policy implementations is the difficulty of representing isolation: A but not B; either RED or BLACK but not both. Representation of such isolation policies as a boolean lattice is straightforward. The objection that the lattice itself includes nonsensical points (such as RED **and** BLACK) misses the point that the policy being embedded in the lattice does not have to include all the points of the lattice. In the specification and implementation of a separation policy, the rules of operation should work to prohibit the aggregation of data that is to be isolated. In fact, the prospect of needing to support isolation policies may make an abstract logic approach especially attractive, allowing the implementation to take advantage of sparse-matrix-like economies.

The provision of useful and practical tools for policy visibility of more than one type could entail significant complexity. Most of the lattice isomorphisms are full of interesting detail and some of them are not (directly) constructive. As a result, the policy conversion code (which will have to be inside a Trusted Computing Base) could become intricate and possibly of some size. As usual, the existence of an isomorphism doesn't promise an easy job of implementation.

CONCLUSION

The diversity of representation and definition for Boolean lattices provides the opportunity for similar diversity in both the policies that can be supported and in the implementation schemes that can be employed. Because of the lattice characterizations, a particular implementation base can be made to match the natural mode of expression of several different-seeming policies through the provision of hidden isomorphism conversions. This conceptual possibility of being able, for example, to support any policy that can be expressed via naive logic with **AND**, **OR**, and **NOT** poses the issue of supplying a far greater number of "categories" than in recommended in [TCSEC85].

BIBLIOGRAPHY

[BIRK48] Birkhoff, Garrett. **Lattice Theory** (1st ed.) American Mathematical Society: Ann Arbor, Michigan, 1948.

[BIBA70] Birkhoff, Garrett, and Thomas C. Bartee. **Modern Applied Algebra.** McGraw-Hill: New York, 1970.

[BIML65] Birkhoff, Garrett, and Saunders MacLane. **A Survey of Modern Algebra.** The Macmillan Company: New York, 1965.

[CLWI87] Clark, D.D., and D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies", *Proc.* 1987 IEEE Symp. on Security and Privacy, 27-29 April, 1987, Oakland, CA, 184-194.

[DENN76] Denning, Dorothy E. "A lattice model of secure information flow," *Comm.* ACM 19, 5, (May 1976), 236-243.

[FLIN87] Flink, W., oral presentation, Third Aerospace Computer Security Conference, Orlando, FL, 7-11 December 1987.

[LEE88] Lee, Theodore M.P., "Using Mandatory Integrity to Enforce 'Commercial' Security", *Proc.* 1988 IEEE Symp. on Security and Privacy, 18-21 April, 1988, Oakland, CA, 140-146.

[LIPN82] Lipner, Steven B., "Non-Discretionary Controls for Commercial Applications," *Proc.* 1982 IEEE Symp. on Security and Privacy, 26-28 April, 1982, Oakland, CA, 2-10.

[SHOC82] Shockley, W. R., "Implementing the Clark/Wilson Integrity Policy Using Current Technology," *Proc.* 11th National Computer Security Conference, 17-20 October, 1987, Baltimore, MD, 29-37.

Proc. 11th National Computer Security Conference, 17-20 October, 1987, Baltimore, MD, 29-37.

[TCSEC83] *Department of Defense Trusted Computer System Evaluation Criteria*, CSC-STD-001-83, 15 August 1983.

[TCSEC85] *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, December 1985.

Appendix. Lattice Theory Results

In this appendix, a set of definitions and results from lattice theory are presented.

Definition L.1: A lattice is a partially ordered set L any two of whose elements x and y have a "meet" $x \cap y$ and a "join" $x \cup y$. [BIRK48, p. 16] [8]

A useful concept in discussing lattices with this definition is that of a "cover" with respect to a partial order. An element a "covers" b provided $a \geq b$ and there is no element x such that $a > x > b$. It is immediate that if the set L is finite, then the partial order is itself characterized by the covering relation.

This first approach of defining a lattice in terms of partial order, meet, and join, however, is not the only way to characterize a lattice.

Theorem L.2: The identities (1) — (4) completely characterize lattices:

(1) $x \cap x = x$ and $x \cup x = x$,
(2) $x \cap y = y \cap x$ and $x \cup y = y \cup x$,
(3) $x \cap (y \cap z) = (x \cap y) \cap z$ and
$x \cup (y \cup z) = (x \cup y) \cup z$,
(4) $x \cap (x \cup y) = x$ and $x \cup (x \cap y) = x$. [BIRK48 p. 18]

The proof of this theorem provides a definition of a partial order $x \geq y$ as the condition $x \cap y = y$. Thus this result shows that a lattice can be defined in terms of meet and join alone and the identities (1) — (4).

Definition L.3: A lattice L is called distributive if and only if, for every $x, y, z \in L$,

$x \cap (y \cup z) = (x \cap y) \cup (x \cap z)$ and

$x \cup (y \cap z) = (x \cup y) \cup (x \cap z)$.

Definition L.4: A lattice L is complemented provided it has both an O and an I [9] and for every $x \in L$, there is $y \in L$ such that $x \cap y = O$ and $x \cup y = I$. y is called the complement of x.

Definition L.5: A lattice L is distributive if and only if it satisfies (5a), (5b) and (5c) identically. [BIRK48, p. 133]

(5a) $(x \cap y) \cup (y \cap z) \cup (z \cap x) = (x \cup y) \cap (y \cup z) \cap (z \cup x)$;
(5b) $x \cap (y \cup z) = (x \cap y) \cup (x \cap z)$;
(5c) $x \cup (y \cap z) = (x \cup y) \cap (x \cup z)$.

[8] A partial order on the set L is a relation between elements that is reflexive, transitive, and antisymmetric.

[9] The element I is the lattice maximum element of the lattice and the element O is the minimum element.

Theorem L.6: Each of the identities (5a), (5b), and (5c) implies (6) below, as well as the other two [BIRK48 p. 133]

(6) If $z \geq x$, then $x \cup (y \cap z) = (x \cup y) \cap z$.

Theorem L.7: Any algebraic system which satisfies

- $a \cap a = a$ for all a,
- $a \cup I = I \cup a = I$ for some I and all a,
- $a \cap I = I \cap a = a$ for some I and all a,
- $a \cap (b \cup c) = (a \cap b) \cup (a \cap c)$ and

  $(b \cup c) \cap a = (b \cap a) \cup (c \cap a)$ for all a, b, c

is a distributive lattice with I.

A final characterization begins with the following interesting ternary operation that arises in the proof of Theorem L.7:

$$(a, b, c) = (a \cap b) \cup (b \cap c) \cup (c \cap a) = (a \cup b) \cap (b \cup c) \cap (c \cup a).$$

Theorem L.8: Let A be any algebraic system with a ternary operation (a, b, c), and elements O, I, such that

- $(O, a, I) = a$,
- $(a, b, a) = a$,
- $(a, b, c) = (b, a, c) = (b, c, a)$
- $((a, b, c), d, e) = ((a, d, e), b, (c, d, e))$,

identically.

Then defining $a \cup b = (a, I, b)$ and $a \cap (a, O, b)$, A is a distributive lattice in which

$(a, b, c) = (a \cap b) \cup (b \cap c) \cup (c \cap a) = (a \cup b) \cap (b \cup c) \cap (c \cup a)$ holds. [BIRK48, p. 137]

Theorem L.9: Every finite distributive lattice is isomorphic to a ring of sets. [10] Assuming the Axiom of Choice, all distributive lattices are isomorphic to a ring of sets. [BIML65]

Definition L.10: A Boolean lattice is a lattice that has O and I and is both distributive and complemented. [BIRK48]

Theorem L.11: Every finite Boolean algebra L is isomorphic to 2^n for some positive integer n. [BIBA70, p. 278]

[10] A ring of sets is a collection of sets closed under union and intersection.

In the proof of theorem L.11, it is shown that the elements corresponding to the set of size n is the set of atoms of L. Atoms are elements that cover O. This parallels the result that a distributive lattice is generated by its meet-irreducible (dually, its join-irreducible) elements.

Definition L.12: An element a of a modular lattice (that is, a lattice satisfying condition (6) of Theorem L.6) is called "join-irreducible" if $a = x \cup y$ implies $x = a$ or $y = a$. Meet-irreducible is defined dually. [BIRK48, p. 20]

Theorem L.13: In a distributive lattice L which satisfies the descending chain condition, each element has one and only one representation as an irredundant join of join-irreducible elements. And dually, if L satisfies the ascending chain condition. [BIRK48, p. 142.]

The Role of "System Build" in Trusted Embedded Systems

J.P. Alstad, C.M. Brophy
Hughes Aircraft Company
Radar Systems Group
213-334-2197

T.C. Vickers Benzel, M.M. Bernstein, R.J. Feiertag
Trusted Information Systems Inc.
213-477-5828

Abstract

We propose a three phase life cycle model for the development of trusted embedded computer systems. We call the middle phase ***System Build.*** First, we propose a definition for embedded systems and distinguish them from traditional multi-purpose computer systems. We summarize the traditional life cycle model, with its development and operational phases, and point out its problems of flexibility and performance for embedded computer systems. Then we introduce the three phase life-cycle model. We describe how the System Build phase allows per-mission software and security configuration and checks security policy offline, thereby allowing a speedup of runtime rights checking, thereby providing increased flexibility and performance.

1 Introduction

There is a growing need for trusted embedded systems to meet critical missions in the DOD. Early attempts to apply trust requirements such as those defined in the Trusted Computer System Evaluation Criteria (TCSEC)[1] or the Trusted Network Interpretation (TNI)[3] indicate that trust requirements for embedded systems go beyond those specified for multi-purpose trusted systems.

Embedded systems must meet stringent performance, minimal complexity and fault-tolerance requirements in addition to computer security requirements. The interplay of trust requirements and mission critical requirements pose special challenges in the development of trusted embedded systems. Quite often the two sets of requirements are in direct conflict. Automated software access control introduces some degree of performance penalty into a system which is straining to meet its performance requirements. Space and complexity factors often make it impractical to strictly meet TCSEC labeling requirements. Embedded systems are often tactical in nature and thus TCSEC requirements applying to Discretionary Access Control, Login, and Identification/Authentication, are often not met. Finally, operational considerations often require interpretation of TCSEC requirements for Operators and Security Administrators, Trusted Facility Management and accompanying documentation.

At first many may conclude that building trusted embedded systems which satisfy their application mandated performance requirements is not realizable with today's technology. To attack this problem we at Hughes and Trusted Information Systems have adopted a three-phase life cycle for embedded systems based on a novel trusted *System Build* phase and associated tools. This use of the System Build phase allows the deployed embedded system to provide a trusted computing base and satisfy security requirements without adversely affecting performance.

This paper begins by providing a definition of embedded systems in Section 2. Then Section 3 discusses the traditional two phase life-cycle of multi-purpose systems. Section 4 describes the "System Build" concept and the three phase development model. Finally, Section 5 presents conclusions and describe the advantages of this approach.

2 Embedded Systems Definition

An embedded computer system or embedded system is a component of a larger system that serves a particular purpose or fulfills a particular application. Its purpose is to provide other than general purpose computing facilities. In fact, while the system is performing its mission no programming is taking place. In practice, there are classes of embedded computer systems whose properties are sufficiently different from general purpose systems as to make them worth considering in their own right. Examples of such embedded systems include process control, battle management, navigation, inventory control, tracking, C3I, countermeasures, and order entry.

Our analysis is focused on the class of embedded systems used in control systems and more specifically, avionics systems. The primary purpose of an avionics embedded system is to assist the pilot in controlling the aircraft. However, an avionics system may also participate in navigation, weapons control, target tracking, communications, life support, as well as other functions. Therefore, an avionics embedded system can be quite complex.

While our work has focused on avionics embedded systems, we believe that many of the properties of such systems can be generalized to a wider class of tactical embedded systems. Thus, for remainder of this report we will use the general term embedded system.

2.1 Differences Between an Embedded System and a Multi-Purpose System

An embedded system is inherently different from a multi-purpose operating system in several essential ways including build cycle, user, operator and administrator roles, and its potential for operational variation.

A multi-purpose system provides a broad support base for a variety of users and applications. Generally, users are grouped into separate roles including administrator, operator, and simply "user". In some systems, the simple "user" role is refined into more detailed categories describing their sophistication and access to different application environments. The applications available within a multi-purpose system span a wide range and may be anything from programming-oriented tools such as compilers and debuggers, to applications such as image processing, order-entry, billing, or mail and communication systems.

While multi-purpose trusted systems are clearly necessary, the life cycle model used in their development has shown limited success when applied to embedded systems, especially those that are mission-critical in nature. If we consider the underlying uniqueness of a mission-critical embedded system, we will discover an alternative design paradigm that can yield better results.

As noted in the list above, an embedded system is not multi-purpose in nature. An embedded system has very specific, limited functions that must be performed in a time critical manner. The I/O interfaces are also static, that is, not added or removed from the system during operation. Human interfaces, if present, are highly stylized and focused on attaining the mission-specific goal.

In addition, an embedded system doesn't generally support the concept of multiple simultaneous users. Autonomous (zero user) systems, such as planetary probes, or military drones are common. When a user is supported in an embedded system, his or her thought processing and reaction time is at a premium. The overhead of login, labeled output, and trusted path can be far outweighed by the necessity to react appropriately within a life-threatening situation.

Another significant aspect of a mission-critical embedded system is the notion of mission-oriented. Such a system is "used" or deployed on many different occasions, but the details of each use can vary. The parameters that distinguish these uses are not known at development time, but can only be determined at mission deployment.

2.2 Defining "User Roles" in Embedded Systems

An important trust distinction between multi-purpose systems and embedded systems, is in the definition of users and personnel in each phase. In a traditional multi-purpose development the personnel in the first phase are the developers, and the personnel in the deployment phase are the end-users which correspond to the notion of "user" found in the TCSEC. In an embedded system, the personnel in the first phase are the developers, the personnel in the middle *System Build* phase are the System Administrators (in the TCSEC sense), and the personnel in the deployment phase, come from a very restricted set of "users" of the embedded system. Furthermore, these restricted "users" do not have many of the trust characteristics traditionally associated with "users" in the TCSEC sense. In fact, many TCSEC requirements pertaining to "users" (Identification, Authentication, Trusted Path, and DAC) are satisfied through a combination of physical and procedural mechanisms in the System Build phase.

3 Traditional Development Model

The traditional software development model, shown in Figure 1, consists of a development effort followed by the operation and maintenance of the software. The software development project usually consists of many activities or phases, such as requirements specification, architecture specification, design, coding, testing, and maintenance. These steps are all aimed at producing a complete system which can be operated independent of the development process. Once the software development is accomplished the software passes into an operational phase where no new development is done and only routine maintenance occurs. This two phase nature of developing software using one set of requirements and operating it in a separate environment is reflected in the TCSEC requirements which distinguish between configuration management during development and trusted facility management for the operational system.

3.1 Problems With The Traditional Development Model

The traditional development model applied to embedded systems would suggest that all applications are fixed at deployment. However, during a mission, an embedded computer typically needs access to information such as data tables which are specific to that mission. When can this information be specified? This mission specific information will be unknown to software developers. On the other hand, the embedded system user may not know this information; in fact, she or he may be expecting to get the information from the embedded computer.

Another flexibility problem concerns minor variations in ranges of configurations. Different missions may require minor variations in types of I/O devices connected to the embedded computer, or the sensitivity of application programs. Again, neither the software developers nor the embedded system user is appropriate to specify the system configuration.

A second type of problem with the traditional development model is system performance. The traditional model leaves much specific rights checking until the software runs. For example, an access request may require identifying the subject, identifying the object, determining the security level for each, checking the mandatory access policy, determining discretionary access information for subject and object, and applying the discretionary access policy rules. Following such a generalized algorithm means that rights checking is probably taking more processing time than is necessary. In a real-world embedded system, where microseconds are precious, such unnecessary processing may lead to unacceptable system performance.

Another performance problem involves user authentication. When the embedded system has a single user, and the entire mission phase consists of one contact between the user and the embedded system, traditional login-based approaches to user authentication will tend to use unnecessary system resources. For example, the extra memory needed to keep password verification code and data should be avoided. We may also mention in passing that in some applications it may be undesirable to spend the time it takes to log on, especially given that the user may be under stress.

4 System Build and The Three Phase Development Model

4.1 The System Build Approach

We at Hughes Radar Systems Group and Trusted Information Systems are attacking the problems of trust, flexibility, and performance by proposing a three phase development model incorporating a new, middle phase known as System Build. During the System Build phase a Security Administrator uses the executable software created during the development phase to produce a load image containing software, mission data tables, and the Access Control Table specific to that mission.

The development phase is similar to the traditional model; however, the trusted System Build tool is also produced. The operational phase (also called the "mission phase") is largely the same as in a multi-purpose systems (except as discussed in Section 2.1). However, this phase does not begin until the actual time of mission definition.

An overview of the three phase development model is given in Figure 2.

Specifically, the load image produced in the System Build phase includes these items:

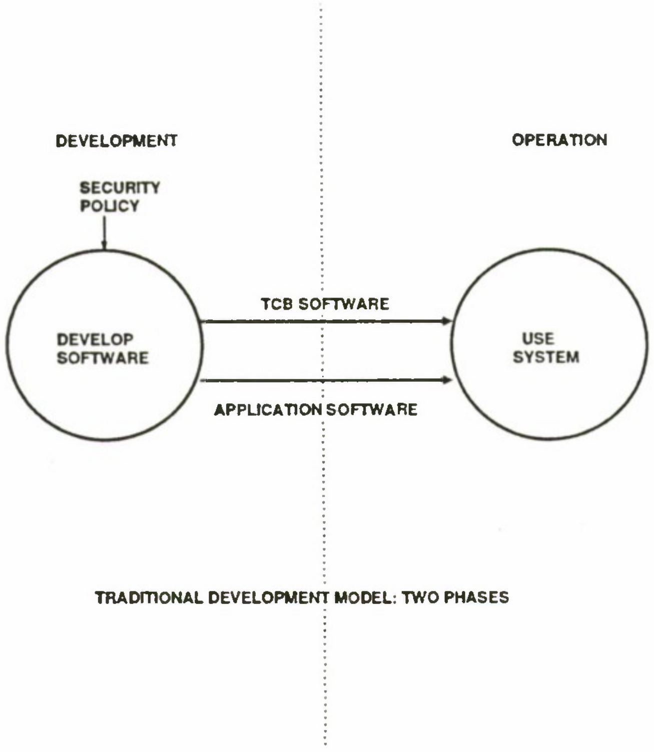


Figure 1: Traditional Development Model

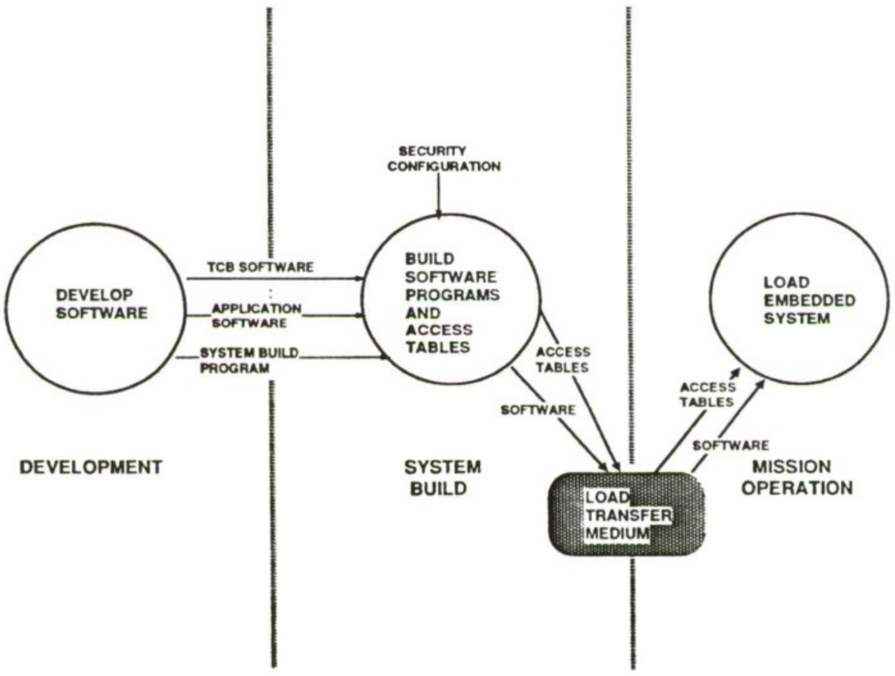


Figure 2: Three Phase Development Model

- The data tables and the software needed for the specific mission. The System Build Security Administrator can exclude unnecessary software and data files. Furthermore, files which are included can contain mission-specific data.
- An Access Control Table which allows efficient rights checking. During the mission phase, rights can be checked with a fast table look-up based on subject id. The comparative dominance relation computation is not necessary having been pre-computed at system build time during generation of the Access Control Table.

The System Build process is described in more detail in following sections.

4.2 Properties Of Embedded Systems Needed For System Build

The System Build approach is designed for embedded systems having a requirement for high performance. In addition, systems using the System Build approach as described in this paper need to have these properties:

- Access control rights are static. That is, subjects and objects are neither created nor removed, and access rights for a subject to an object[1] are unchanging during a mission.

  Since the software environment in a typical embedded system is static (See Properties 3 and 4 in section 2.1), these properties should normally be easily achieved.

  Relaxing this restriction is discussed in section 4.6.
- The system has exactly one "user" per mission (See Property 1 in section 2.1).
- No programming is done on the embedded system during the mission (See Property 6 in section 2.1).
- All external devices are single-level. In particular, any device producing output for the system user can be assigned the user's security level (See Property 5 in section 2.1). Since the user's interactions with the system are restricted and well defined, this should not be a problem.

While not all embedded systems have all the above properties, we believe that the System Build approach is widely applicable.

4.3 Philosophy Of Protection Supporting System Build

The System Build philosophy of protection includes these points:

1. The objects in the embedded system include files, message classes, etc.
2. The subjects are programs and external devices. (The human user is not a subject. Rather the specialized input/output devices he or she uses are included as subjects.)

[1] For simplicity, only subject/object entries in the Access Control Table are discussed in the paper because the same considerations apply to subject/subject entries.

3. The rights for subject/object accesses in the embedded system are kept in an Access Control Table. (The Access Control Table is logically a two dimensional array, with the rows representing subjects, the columns representing objects, and the entry in a particular row and column representing the access rights of that subject to that object.) The Access Control Table is created offline by the System Build program and used by the TCB during the mission phase.
4. The Access Control Table is constant during the mission phase. This is possible because the access rights are static as discussed in section 4.1.

Because the Access Control Table is static, it is possible to check whether any Access Control Table is "secure"; i.e., is in agreement with the embedded system's security policy. Consequently during the mission phase a subject/object access which is allowed by a secure Access Control Table is in agreement with the security policy.

This implies that during the mission phase the TCB only needs to determine subject and object identities to check access rights; in particular, the TCB does not need to compare security levels against a dominance relation (i.e., the TCB performs a simple table look-up instead of computing a point on a lattice). This typically allows the TCB to execute faster.

4.4 The System Build Process

The System Build program is run by the Security Administrator (or a designee) on a trusted offline computer, typically once per mission.

There are three inputs to a System Build run:

1. The software to be loaded into the embedded computer. This would include the Trusted Kernel, untrusted parts of the operating system, trusted application software[2], and untrusted applications. All of this is in executable form.
2. The Security Configuration. This is more or less the information in the Access Control Table. The Security Configuration includes:
   - Identification, security level, and discretionary access information for each security subject.
   - Identification, security level, and discretionary access information for each security object.
   - Designation of trusted subjects.
   - All authorized direct accesses, giving subject, object, and type of access for each.
   - Any information required to support a denial of service policy, such as resource usage limits for each program.
3. Mission-specific data files.

Output from the System Build program is an image containing the input software, the Access Control Table determined by the Security Configuration, and the mission-specific data files. Output is written to an appropriate medium for later loading into the embedded computer.

[2]Example trusted application software includes system management programs and message downgraders.

The System Build program checks that the input obeys the security rules; if not, it does not generate an image. The security rules include these:

- Each subject identifier must be unique; likewise for each object identifier.
- Message classes going to or from an external device take on the security level of the external device.
- The input subject/object access rights must be in concurrence with the security policy. The security policy will typically include rules like the simple security condition and the *-property.

Each of these is a simple check. In particular, checking that the input access rights obey the security policy is simple due to the static nature of the access rights as discussed in section 4.2. (But see section 4.4.)

It can be seen that using the System Build approach means that the security policy is applied at System Build time, while the mission time check is a simple, fast table access. This is analogous to the situation of hardware supported secure file reading. When a file is opened, a slow, software check is made to see that the requesting program is allowed to read that file; if it is, the hardware is set up to allow reading from the file, but only to the addresses owned by that program. Then when data transfers occur, a fast, hardware check is made to see that the transfer is to a legal address. In both cases, the slow, software policy decision is made only once, while repeated checking is done in a simple, fast manner supported by hardware.

4.5 Identification and Authentication Through System Build

Trusted embedded systems are often intended for use in a tactical or combat arena. In these environments it is not practical to require that the single "user" of the embedded system go through the process of automated login or use a trusted path. Nonetheless, the TCSEC requirements pertaining to Identification, Authentication, and Trusted Path are still significant. We believe that the System Build facility provides the mechanism to perform Identification, Authentication and Trusted Path prior to mission deployment where it is necessary for the "user" to react appropriately within a life-threatening situation.

In the the System Build approach, the result of the build process is a software program which is user specific and usually also processor specific. The single user has exclusive possession of both the medium containing the TCB and other software and control of the embedded system itself. In this case, the fact that the TCB is loaded in the system provides both identification and authentication of the user.

The user's actions in physically transporting the software to the system and loading it take on the properties of the trusted path.

As described above, System Build creates an image for loading into the embedded computer. Since this image can be created on a mission by mission basis, it is possible to specify a build configuration on a per "user" basis. Then "user" Identification can be carried out through physical and procedural means involving a person identifying him or herself to the System Build Administrator. At this point then the "user" is authenticated to receive the image. At this point, the "user" and the image medium now function as a trusted path to the embedded system where it is loaded and brought to an operational state.

4.6 System Build With Non-Static Access Rights

As described above, the System Build philosophy of protection is most effective if access rights are static. However, the System Build approach is applicable in some cases where access rights are not static.

The motivation for System Build is to allow checking of access rights based only on subject and object identity (and the current contents of the Access Control Table). Therefore the System Build approach can be very effective even with non-static access rights providing these two properties hold:

1. Access rights, rights to change access rights, can all be represented in an Access Control Table.
2. For any such Access Control Table, it can be determined whether or not the security policy will always be maintained given that Access Control Table as initial state.

Performing the computation to check the security of the Access Control Table, may require large amounts of computer time. However, the offline nature of System Build may mean that the computer time is available; in realistic embedded systems, it may be worthwhile to spend hours during System Build to save micro-seconds during the mission phase.

A more exact characterization of the situations where the System Build approach is appropriate with non-static access rights is a subject for future study.

5 Conclusions

Mission-oriented embedded systems necessitate changing TCB's to satisfy mission specific requirements. However, modifying TCB's invalidates a TCSEC rating. We believe that by defining embedded system *System Build* as part of the TCB and evaluating its trust characteristics it will be possible to satisfy certain TCSEC requirements, and provide trusted mechanisms for building mission specific trusted embedded systems.

The use of a middle System Build phase provides significant advantages in the design and development of trusted embedded systems. We believe that the use of a System Build phase can increases the mission phase performance of trusted embedded systems. Further, we believe that there are high performance trusted embedded systems which are infeasible without the use of System Build. The approach presented in this paper provides three key advantages over traditional methods of developing trusted systems.

- It is possible for trusted embedded systems to satisfy TCSEC requirements(Identification, Authentication, DAC) which otherwise they might not be able to satisfy.
- The use of a System Build phase allows increased performance of the mission phase.
- System Build provides flexibility so that costly, complex embedded systems can be configured in a trusted manner to meet mission needs.

Acknowledgments

Several people have contributed to the ideas developed in this paper. Glenn Ladd of Hughes suggested the concept of the mission phase as a static system with minimal users. Jeff Wagner of Hughes recognized the need for pre-allocation of access rights, and Gary Miyahara of Hughes developed the operating system concept in support of this. Marv Schaefer of Trusted Information Systems and Jerry Popek of Locus recognized that establishing the security of the static access matrix could be accomplished in a manner similar to that of the Trusted Kernelized VM/370[2].

References

[1] "Department of Defense Trusted Computer System Evaluation Criteria," DOD 5200.28-STD, December 1985.

[2] Schaefer, M. et al., "Program Confinement in KVM/370", Proc. 1977 ACM Annual Conf., p. 404-410.

[3] "Trusted Network Interpretation of The Trusted Computer System Evaluation Criteria", NCSC-TG-005, Version-1, July 1987.

COMBINING SECURITY, EMBEDDED SYSTEMS, AND ADA PUTS THE EMPHASIS ON THE RTE: ARTEWG ESTABLISHES A SECURITY TASK FORCE

Fred A. Maymir-Ducharme, Ph.D
IIT Research Institute
Lanham, MD 20706
(301) 459-3711

David Preston, Ph.D
Catholic University
Dept. of Computer Science
Washington D.C. 20011
(202) 635-5193

Mary Armstrong
IIT Research Institute
Lanham, MD 20706
(301) 459-3711

INTRODUCTION

The security, embedded systems, and Ada language domains have never been unified. Systems soon to be implemented, such as the Air Force Advanced Tactical Fighter (ATF), are now forcing the development of integrated solutions to concerns in these areas. This paper will describe issues common to the three domains, identify the groups addressing them, and detail the work of the Ada RunTime Environment Working Group (ARTEWG) Security Task Force, whose charter is to focus exclusively on these issues.

ISSUES RELATED TO SECURITY, EMBEDDED SYSTEMS, AND ADA

Three previously distinct domains, security, embedded systems, and the Ada language, are rapidly becoming integrated. Before secure embedded systems can be implemented in Ada, many issues must be resolved.

Traditionally, the domain of secure systems within the Department of Defense (DoD) was limited to information processing systems, commonly referred to as automatic data processing (ADP) systems. The concept of security in these systems was limited to the prevention of compromise (e.g., nondisclosure of sensitive or classified information). The concept of security is expanding to include the preservation of integrity and the assurance of service. Secure systems will soon be expected to prevent compromise, preserve integrity, and assure service.

This expanding concept of security is particularly challenging for embedded systems. In an embedded system, the computer or processor is just one of many hardware components. The primary objective of the system may be, for example, to control a weapons system or provide navigation support for an aircraft. Data processing is a means to support the objective, not an end in itself. Therefore, the data processing component of the system must support the system objective and be consistent with the system's requirements.

Another attribute of embedded systems is, typically, their time-critical nature. Processing speeds must be consistent with system requirements to respond to external conditions. Inputs from radar, for instance, must be processed fast enough for the system to respond to a minimum number of the radar signals received.

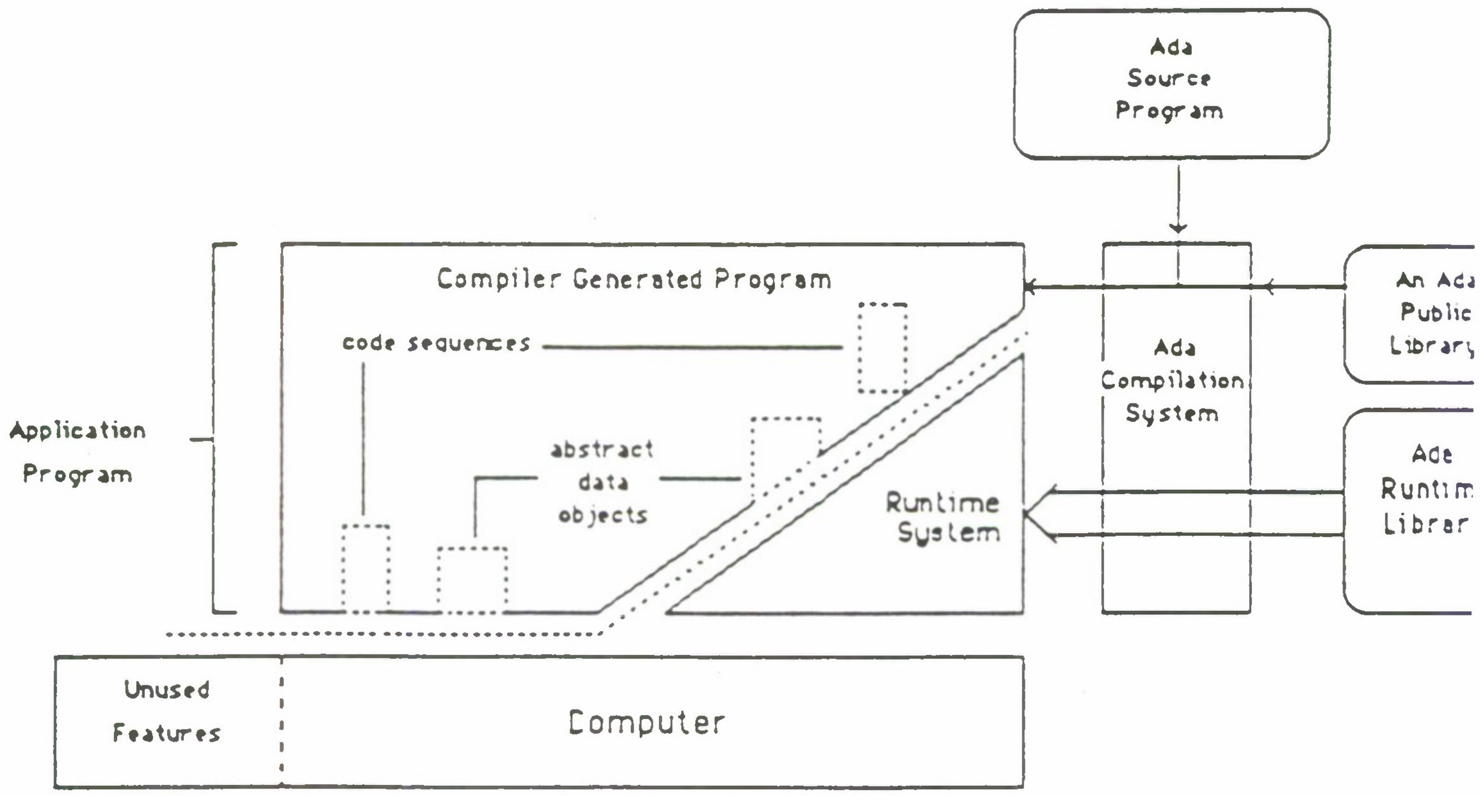
Time, therefore, is a valuable and limited resource. For similar reasons, throughput is frequently near system capacity. The burden of security is imposed on these systems, which are already operating with limited resources.

Embedded systems often function in the "system high" mode to avoid the overhead of supporting multi-level security. This "system high" approach is unsatisfactory for the embedded systems currently being planned. As a result, these systems must now meet greater security demands with fewer resources than were available to their information processing predecessors.

The Ada programming language is the third component of today's military systems. DoD Directives 3405.1 and 3405.2 require the use of Ada for embedded and other DoD systems. Many of Ada's features directly support the implementation of security mechanisms.

However, Ada is new to the security community, a community which favors languages and compilers with a well established track record. In addition to being new, the Ada RunTime Environment (RTE) presents unique concerns.

Since embedded systems have no operating system, they have traditionally relied on an application-specific runtime executive to provide a limited set of operating system functions. The Ada RTE is generated by the compiler to provide this runtime support. With the increasing maturity of Ada compilers, there is increasing sophistication in the generation of RTEs. If, for example, a program does not use the concurrency mechanism of Ada, the compiler may not generate the portion of the RTE which supports concurrency. In this way each program compiled may generate a slightly different RTE. Please refer to the diagram below for a system view.



In merging the domains of security, embedded systems, and Ada, the RTE has become the primary focal point for several reasons. Although security mechanisms will be written in Ada, it is the RTE which will support the execution of the code. Gaining assurance that mechanisms are coded properly is necessary; but so is gaining assurance that the code will be executed as intended. Further, embedded systems typically have real-time requirements. Therefore, developers of compilers for these systems stress efficient implementation. The RTEs must be optimized. If security mechanisms are to be efficiently implemented, they must drive and exploit these RTE optimizations. Direct support of security mechanisms by the RTE may be the only viable way to simultaneously satisfy both security and timing requirements. Perhaps the largest question of the RTE is related to compilers producing different RTEs for different compiled programs. If security mechanisms are built into the RTE, then assuring that altering the RTE will not have a detrimental effect on the security mechanisms will be very challenging.

GROUPS ADDRESSING THESE ISSUES

Several government agencies have addressed various elements of secure, embedded systems in Ada. The National Computer Security Center (NCSC) has been investigating these issues since the mid-1980s. Their funded research has included assessing the viability of applying formal verification techniques to Ada, developing guidelines for the use of Ada on secure systems, and exploring an interpretation of the Trusted Computer System Evaluation Criteria (TCSEC) for embedded systems. The Defense Advanced Research Projects Agency (DARPA) has also funded research in formal verification of Ada software as well as in defining a process model for the development of trusted systems. The Rome Air Development Center (RADC) has also been involved in research in formal verification of Ada software and has researched software development methods for trusted systems.

A joint industry-government working group is approaching these issues from a more pragmatic perspective because their implementation must meet these requirements. The Joint Integrated Avionics Working Group (JIAWG) consists of government personnel responsible for three major avionics systems in various stages of development, and representatives from the contractors supporting those systems. The systems are the Air Force Advanced Tactical Fighter (ATF), the Army Light Helicopter- Experimental (LHX), and the Navy Advanced Tactical Aircraft (A12). Each of these systems is an embedded system; each will have security requirements, and each is to be coded in Ada. To support the JIAWG, AdaJUG (Ada Joint Users Group) established the Common Ada RunTime (CART) requirements for a common RTE for the JIAWG applications. These requirements will include security requirements.

The Ada 9X Project is currently managing the revision of the Ada programming language, ANSI/MIL-STD-1815A. The revisions are being accomplished by several teams, including designers, users, and implementers. In addition to the existing teams (Distinguished Reviewers, the Requirements Team, the Mapping / Revision Team, etc.), the Air Force Armament Laboratory (AFATL/FXG) is forming the "Language Precision Team" (LPT). The LPT will address the security oriented and safety critical systems requirements for the Ada 9X project. This team will be contracted to provide the formal definition of various Ada language features such as the set of optimizations allowed by the Ada Reference Manual Chapter 11.6, and the formal Ada tasking state-transition model. These formal specifications are necessary in order to support highly predictable and reliable software.

The primary volunteer organization to investigate Ada RTE issues is the ACM (Association for Computing Machinery) SIGAda (Special Interest Group - Ada) Ada RunTime Environment Working Group (ARTEWG). This group has recently established a task force chartered to identify and address security issues relative to the Ada RTE - the ARTEWG Security Task Force. ARTEWG and the Security Task Force include individuals from government, industry, and academia.

RELATED PAST WORK

The Security Task Force will draw on several previous efforts as a basis for its work. ARTEWG has published several documents describing Ada runtime environments:

- Catalogue of Ada Runtime Implementation Dependencies
- A Framework for Describing Ada Runtime Environments
- First Annual Survey of Mission Critical Application Requirements
- Catalogue of Interface Features and Options for the Ada Runtime Environment (CIFO)
- A Model Runtime System Interface

These documents describe the requirements for RTEs, their components, potential differences when implemented with Ada features, and interfaces between RTEs and applications. ARTEWG has also proposed that an Ada Runtime Dependencies Guide be developed. This document is intended to identify and clarify aspects of the Ada language that are dependent on the implementation of the runtime environment and to provide guidance on the use of such implementation dependent Ada features.

The National Computer Security Center (NCSC) has funded several studies to examine software for trusted systems, most recently the "Study of the Use of Ada in Trusted Computing Bases (TCBs) to be Certified At Or Below the B3 Level." Ada offers various specific benefits for the development of TCBs, such as strong data typing facilities, information hiding with the use of packages, and the capability to create TCB systems that exhibit modularity. This study maps the Department of Defense Trusted Computer System Evaluation Criteria (TCSEC) (DoD 5200.28-STD) to the software development process. It also provides programming guidelines for developing Ada software for TCBs.

The three JIAWG applications, ATF, A12, and LHX, are each scheduled to be coded in Ada. The JIAWG Security Policies are, therefore, documents that apply to secure, embedded systems to be developed in Ada. The documents are, however, language independent. Two other documents, each currently in draft form, will directly address and affect security. The JIAWG common avionics architecture document will define bus bandwidths and other architecture characteristics that will affect such security concerns as the system's ability to support sensitivity labels. The JIAWG software engineering environment (SEE) requirements document will define requirements which will determine the extent to which the SEE can provide automated support of security mechanisms during the development and maintenance of the JIAWG platforms.

The "Workshop on Issues of Integrity and Security in an Ada Runtime Environment" was held on April 3-5, 1990, in Orlando, Florida. The workshop was co-sponsored by IIT Research Institute (IITRI), the University of Houston - Clear Lake, and the Ada Joint Program Office (AJPO). The workshop brought together specialists from both, the Ada and security communities. The attendees defined two goals. The first goal was to identify and discuss the security and integrity issues related to an Ada runtime environment, and the second was to create some synergy between the two groups in order to address these issues thoroughly and to establish the communications necessary for future work in this area.

The participants in the workshop were divided into three working groups: the Ada Runtime Working Group, the Access Controls in Distributed Environments Working Group, and the Formal Methods Working Group.

Ada Runtime

The Ada Runtime Working Group focused on the security and integrity issues (this paper uses the phrase "security and integrity" to represent the three security mandates: prevention of compromise, preservation of integrity, and assurance of service) that are a result of Ada RTEs. This working group was chaired by Ms. Dock Allen of Control Data and Mr. Richard Powers of Texas Instruments. The group addressed the following issues: the identification of general threat types; the definition of a working model of the Ada RTE and its interfaces; the analysis of security issues for typical Ada runtime features; the allocation of security requirements to a typical Ada runtime; and Ada features required to build integrity into applications. The working group developed a list of functions typically supplied by an RTE (such as scheduling, initializations, and communication between tasks). This list was then used to analyze parts of the Ada RTE (and functions of Ada) that may be considered a risk to security and integrity. The working group concluded with the following recommendations for future work:

- Evaluate the feasibility of using host tools to check programs for secure and high-integrity use of Ada.
- Evaluate ARTEWG's CIFO from a security and integrity perspective.
- Propose and evaluate alternate TCB software architectures.
- Propose and evaluate alternative approaches to subject boundaries (fire-walls).
- Evaluate where current compilers do not efficiently support Ada features (such as dynamic memory management) that are valuable for security and integrity.
- Identify hardware support needed for, or beneficial to, proposed secure software architectures.
- Develop guidelines for the use of Ada in secure, high-integrity systems.
- Examine and recommend approaches for tools to control use of Ada external runtime library (XRTL) features.
- Continue to identify, evaluate and address security-related Ada RTE issues and problems.
- Foster research addressing formal verification of concurrent Ada.
- Develop guidelines for the use of ARTEWG's CIFO with secure systems.

In the interest of security, the group also strongly supports any effort to provide more predictability and formalism for Ada in the Ada 9X Project's revision of the language.

<u>Access Controls in Distributed Environments</u>

The Access Controls in Distributed Environments Working Group, chaired by Dr. Charles McKay of the University of Houston, Clear Lake, addressed the research and development issues necessary to facilitate practical progress in future security projects of wide significance. This group focused on the capabilities that are unlikely to be available in a timely fashion unless these research and development issues are properly addressed; these issues included access control in distributed environments, to include the balance between the functional requirements of a project and the nonfunctional requirements, such as timing and spacial constraints; the semantics of access control in distributed target environments, supporting dynamic, multilevel security and integrity in an incrementally evolving, distributed target environment; issues of a trusted computer base (TCB) that extend across portions of the hardware, the Ada RTE, the extended runtime library (legal extensions such as those proposed in ARTEWG's CIFO), and parts of the application (this included discussion on the fire-walled portions of the applications); and the multidimensional issues involved in the mapping of DMLSI (distributed multi-level security and integrity) concerns to considerations of hardware, software criticality and sensitivity, and time and space. In response to these issues, the working group developed several recommendations:

- Evolve a standard, Conceptual Reference Model (CRM) for runtime environments tasked to support mission and safety-critical applications in distributed environments.
- As the highest priority for the use of the CRM, specify and develop the distributed kernel's interface set. The CRM interface set should support a "single site image"; that is, the distributed nature should be transparent.
- The Federal government should contract for actions ranging from the development of proof-of-concept implementations, validation test suites, etc., to formal models and methods for the distributed kernel and distributed applications.
- Similar government contracts should follow to create new CIFOs for distributed information services, distributed communication services,

distributed configuration-control services, and distributed operating-system services.

Formal Methods

The Formal Methods Working Group, chaired by Dr. John McHugh of Computational Logic, Inc., addressed a variety of issues associated with the Ada code that becomes part of a TCB - regardless of whether this code represents a trusted application, a runtime system (RTS), or an operating-system kernel.

The group agreed that the TCSEC is understood reasonably well; nonetheless, it is not a basis for a true formalization of security. With an increasing tendency towards the formulation of mission-specific security policies and the notion of trusted applications, a more flexible and general framework is appropriate. The group identified a list of research and technology transfer issues:

- What methodologies are suitable for using formal methods to develop and maintain trusted Ada runtime systems? What are the concepts that need to be axiomatized? What is a good formal language to express security and integrity properties? What are the appropriate paradigm and vocabulary? What are appropriate formal methods for security and integrity in Ada? What is a formal language that flows down well into system/software implementation languages such as Ada? What tools are required to support the above methods and methodologies?

- What is the relationship between the RTS and application security and integrity?

- Is there an incremental approach to the development of formalisms, methods, and tools? What useful short-term research results can be obtained through incomplete and/or approximate formalisms? (For instance, how do we handle ambiguous and incomplete runtime models?)

The technology transfer issues listed were:

- How should formal methods be introduced into practice?

- What we can say today about handling with the informality of existing languages, systems, and specifications?

In conclusion, the working group's position was that we should investigate further the use of Ada safe subsets. Work has been done in this area by TRW (ASOS), Odyssey Research Associates (Penelope), Computational Logic, Inc. (AVA), and the National Physical Laboratory, U.K. (Low Ada).

FUTURE DIRECTIONS OF THE ARTEWG SECURITY TASK FORCE

ARTEWG recognizes the industry's need to address security issues from the runtime environment perspective. Sufficient interest and issues were raised at the 1989 Fall ARTEWG meeting to justify the creation of a new task force to address these issues. The newly formed Security Task Force, chaired by Dr. Fred Maymir-Ducharme, formally met for the first time at the Winter 1990 ARTEWG meeting to define the task force charter. The charter states that the task force will concentrate on runtime environment issues germane to security and integrity. The purpose of the task force is to study security issues associated with the Ada RTE and report the findings. This task force will review the output from the other ARTEWG subgroups and task forces, and it will make the necessary recommendations to ensure that security issues have been adequately addressed. Security restrictions, architectures, guidelines, standards and modelling techniques are some of the issues presently addressed, as well as their relationship to the "Orange Book" (TCSEC - DoD 5200.28-STD). The group's first two tasks were identified. The first task is to review the current CIFO entries and identify the associated security issues and concerns. The second is to generate and centralize the following information: current security technologies; models and architectures; and a glossary of security terminology and references. The Security Task Force will produce two documents. The first report will identify and define the relevant RTE security issues. The second report will provide a summary of the research and evaluations done by the task force. The task force will document approaches currently in use, propose security approaches, and provide guidelines for the support of these approaches.

The first two meetings of the Security Task Force resulted in several action items. The group will review the CIFO entries and the relevant Ada 9X Revision Requests (RRs) to identify the security concerns associated with each entry. The resulting reports will be submitted to the ARTEWG group working on the CIFO and to the Ada 9X Project Office. It is also planned that they be published in Ada Letters. Another action item is to establish communication with other groups addressing similar security issues. The CMU group implementing additional features for the MACH operating system and the IEEE group defining the POSIX operating system standards are two such groups. The task force will investigate the status of the POSIX group dealing specifically with security and supply POSIX with the appropriate support and information on Ada RTE security binding issues. The task force also plans to review the "Secure MACH" (aka: T-MACH or Trusted MACH) requirements and supply the necessary feedback to the Navy's Next Generation Computer Resources (NGCR) organization. Several topics of interest to ARTEWG, include:

- interpreting security and trust requirements to implement application systems at the C2 level;
- using formal methods to address integrity and security issues for Ada RTEs;
- resolving requirements for security and optimization;
- architectures for secure Ada runtime support; and
- Low Ada and a trusted Ada kernel.

During 1990, we expect significant progress to be made in defining the intersection of requirements for embedded systems, Ada runtime environments, and security and integrity. Successful completion of this effort requires input from the traditional security perspective, as well as review by the Ada and embedded systems communities.

REFERENCES

"Study of the Use of Ada in Trusted Computing Bases (TCBs) to be Certified at, or Below the B3 Level," National Computer Security Center, prepared by IIT Research Institute, April 1989.

"Ada Verification Systems Study," National Computer Security Center, prepared by IIT Research Institute, April 1989.

Thompson, K., "Reflections on Trusting Trust: Turing Award Lecture," Communications of the ACM Vol. 27, No.4, Aug. 1984.

Rowe, K. and Ferguson, C., "Ada Technology/COMPUSEC Insertion Status Report," Proceedings of the 10th National Computer Security Conference, Sept. 1987.

"Requirements for Developing Trusted Embedded Real-Time Computer Systems," National Computer Security Center, prepared by IIT Research Institute, to be published in 1990.

"Catalogue of Ada Runtime Implementation Dependencies," ARTEWG of ACM SIGAda, Dec. 1, 1987 (version 2.0).

"A Framework for Describing Ada Runtime Environments," ARTEWG of ACM SIGAda, Oct. 15, 1987.

"First Annual Survey of Mission Critical Application Requirements," ARTEWG of ACM SIGAda, Dec. 1, 1987.

"Catalogue of Interface Features and Options for the Ada Runtime Environment, Release 2.0," ARTEWG of ACM SIGAda, Dec. 1, 1987.

"A Model Runtime System Interface 2.3," ARTEWG of ACM SIGAda, Oct. 11, 1988.

DISCLOSURE PROTECTION OF SENSITIVE INFORMATION (Variations on a Theme in C Major)

Ingrid M. Olson
MITRE Corporation
7525 Colshire Dr.
McLean, VA 22102

Eugene F. Troy
National Institute of Standards & Technology
Computer Security Division
Gaithersburg, MD 20899

Milan S. Kuchta
Department of National Defence
System Security Centre
719 Heron Road
Ottawa, Ontario K1G 3Z4

Brian W. McKenney
MITRE Corporation
7525 Colshire Dr.
McLean, VA 22102

INTRODUCTION

Scope and Purpose

This paper presents three approaches to the protection of sensitive unclassified information against unauthorized disclosure, two based on U.S. policy and one based on Canadian policy. An analysis of the approaches is provided based on the differences in how each approach defines hierarchical levels and non-hierarchical sets of sensitive[1] information, and the basis for determining the "trustworthiness" of the users. In addition, the approaches discuss the use of *Trusted Computer System Evaluation Criteria (TCSEC)* trusted technology to meet the confidentiality (non-disclosure) protection requirements.

This paper came about largely due to the fact that there is no comprehensive guidance in effect today that covers the protection of sensitive unclassified information. The authors have all spent considerable time and energy in trying to develop some guidance for different communities of interest (i.e., Federal Government, Department of Defense (DOD), Canadian Government, and private sector) and have had little success in developing uniform protection requirements. It is our hope in presenting this paper with these three proposed approaches, that a framework suitable or adaptable to all communities of interest will emerge.

This paper addresses computer security requirements relating to confidentiality, and does not include requirements relating to integrity or availability. Integrity and availability, however, are at least as important as confidentiality in many applications handling sensitive information. This must be carefully considered when determining the overall computer security requirements of a system. In addition, only computer (i.e., technical) security issues are addressed which can be dealt with by use of trusted systems technology. It is assumed that the appropriate physical, administrative, procedural, emanations, communications, and other related protection measures adequate to the sensitivity of the information being handled are already in place.

U.S. Policy

Numerous policies exist that require U.S. Federal agencies to protect sensitive information. There are two general mandates: (1) Public Law 100-235, *The Computer Security Act of 1987*, which requires that systems processing sensitive information be adequately protected [1], and (2) OMB Circular No. A–130, which establishes requirements for Federal agencies to protect sensitive information [2].

In addition, other statutes, laws and policies exist that require the protection of specific types of information. Much of this information is unclassified information that is exempt from release under the Freedom of Information Act. Other statutes and policies include *The Privacy Act of 1974, The Administrative Procedures Act, Title 18, U.S. Code 1905, The Bank Secrecy Act, The Foreign Corrupt Practices Act*, and DOD Directive 5100.36 (*Defense Scientific and Technical Sensitive Information)*.

[1] In this paper, the term "sensitive" refers to sensitive, unclassified information.

What is Sensitive Unclassified?

Public Law 100-235 defines sensitive information as follows:

> ... any information, the loss, misuse, or unauthorized access to or modification of which could adversely affect the national interest or the conduct of Federal programs, or the privacy to which individuals are entitled under the Privacy Act, but which has not been specifically authorized under criteria established by an Executive order or an Act of Congress to be kept secret in the interest of national defense or foreign policy.

Examples of sensitive information include privacy information, proprietary information, financial information, personnel information, procurement sensitive information, research information, program plans, and contract information.

It appears to be generally accepted that there are various levels and kinds of sensitive information, some of which may require stronger protection mechanisms than some classified information (e.g., information involving extremely large financial sums, critical mission-sensitive information). It is our belief that sensitive information is not part of the same hierarchy (i.e., not on the same lattice) as classified, but is on a number of separate lattices depending on the kind of information and the security domain in which it exists. Within the Federal Government and certainly within the private sector, numerous lattices may exist. The protection of business and financial data crucial to commerce and industry is as important to the national interests as to corporate survival, and requires high levels of protection.

Approaches to Defining Protection Requirements

Establishing and implementing an Information Security (INFOSEC) program involves identifying the sensitivity of information and determining an appropriate level of trustworthiness for individuals accessing the various types of sensitive information. This practice is well understood for the protection of classified information. The DOD has defined user clearance levels and classification guides that assist the information owner in determining the appropriate level of classification for various types of information. The determination of the appropriate sensitivity should include the evaluation of the value of the information both to the organization and to potential unauthorized users. No standards such as clearance and classification currently exist for sensitive information.

In addition to identifying the appropriate information sensitivity, standardized procedures for the marking and handling of sensitive information are needed. The DOD has precise policies outlining accountability, storage, transmission, and destruction requirements for classified information, and similar policies are needed for sensitive information.

THREE PROPOSED APPROACHES

This section describes three proposed approaches for disclosure protection of sensitive information. The NIST approach for protecting sensitive information in Federal government computer systems described below is broader and more general than the proposed DOD or Canadian approach.

NIST Approach

National Institute of Standards and Technology (NIST) guidance applicable to all Federal departments and agencies must be general enough to permit those organizations to develop their own specific implementation approaches. NIST has developed some basic principles for protection of sensitive information including the use of trusted systems technology, and is in the process of expanding these principles into formal guideline documents for Federal agency use. The following information represents an overview of those principles and some conclusions that may be drawn from them.

The *Computer Security Act of 1987* assigns NIST the responsibility for developing security standards and guidelines for unclassified Federal computer systems (except certain special-purpose DOD "Warner Amendment" systems) [1]. NIST is therefore responsible under the Act for advising Federal agencies, DOD among them, on the applicability and use of protective measures, including trusted systems technology, in Federal computer systems that process unclassified information. This includes recommending methods of identifying, marking, and protecting sensitive information resident in computer systems. NIST is also responsible under the Act for assisting the private sector upon request in using and applying the results of the security standards and guidelines program. Accordingly, NIST guidance should be broad enough to be helpful to the private sector as well as to Federal agencies.

Establishing Basis of User Trust

Job-related need has traditionally been the primary basis for permitting access to information sensitive to disclosure. User trustworthiness has been a secondary and supporting requirement satisfied either implicitly or overtly. Within the classified community, trustworthiness has been based overtly on an investigation of some sort leading to a security clearance. Although mechanisms exist to establish analogs of that process in Federal agencies via Office of Personnel Management (OPM) position sensitivity levels, these have not been widely implemented in many agencies. No comparable program exists to any magnitude in the private sector.

Most organizations will continue to depend upon job function as the principal basis for permitting access, with the trust requirement satisfied implicitly by job definition. Access to job-related sensitive material where required is generally considered to be an integral part of the job duties, and failure to adhere to confidentiality requirements for the job can be a basis for adverse action. Therefore, "need-to-know" in the strict job-related sense of that term is the single common basis for trust leading to information access in the civil and private sectors.

Organizations often desire to more formally delineate levels of trust beyond job definition, based on a variety of factors, such as grade level of the employee, years of service, or demonstrated prior trustworthiness. Consideration of such factors in essence constitutes a risk analysis of the likelihood of disclosure by a particular employee. Some organizations, which feel it is warranted by analysis of risk, may choose to adopt the OPM position sensitivity level process.

Partly because there is no generally applicable structure for identifying levels of trustworthiness, the NIST security approach stresses the use of the risk management process to determine adequate safeguards for a particular system. A risk analysis considers system-related assets and vulnerabilities, along with potential threats and their likelihoods, and forms the basis for cost-effective security decisions via the countermeasure selection process.

Information Sensitivity to Disclosure

Four basic principles of information sensitivity to disclosure must be discussed.

1. Some TYPES (also called "categories") of information can be identified that, if disclosed without proper authority, inherently could do harm to the organization, its employees, or others. The Freedom of Information Act encompasses most of those types in its list of matters exempt from disclosure, and other agency-specific types may be identified.

2. Arbitrary LEVELS of sensitivity to disclosure can be established for most of those types of information. Those levels are typically expressed in degrees of potential consequences to the organization's mission or harm to individuals.

3. It is feasible to map between a level of disclosure sensitivity and a set of protection requirements that must be met to protect the information at that level. Operating environments where the information might reside each have their own inherent protection mechanisms and risks which need to be addressed to assure the requirements are met.

4. In the absence of labeling standards, it is difficult to assure a clear mapping of disclosure protection requirements across security domains for information of the same sensitivity type or level. For instance, two security domains called the Federal Bureau of Investigation (FBI) and the Bureau of Indian Affairs (BIA) might both use a sensitivity type called "internal working papers," with little inherent comparability in protection requirements for information assigned the same level. If the FBI were to loan files to the BIA, special effort would be required to assure that the BIA protected the information according to the protection policies of the FBI.

The following example of some arbitrary hierarchical levels of disclosure sensitivity has been suggested to show how agencies could construct their own information categorization schemes and set minimum standards of protection. Three disclosure sensitivity levels (from "low" to "high") appear to be the most useful, plus a "null" level for information releasable to the public. Sample definitions of these levels are as follows:

Agency Level 0 (null) — No special disclosure protection is required (although integrity and availability protection might be needed). No damage due to unauthorized disclosure is anticipated.

Agency Level I (low) — Unauthorized intentional or inadvertent release of information could minimally compromise effectiveness of Department or Agency. Could also include inconvenience to individuals or very minimal privacy violations. Normal protective requirements on multi-user systems for this level are user identification and authentication and personal accountability as a minimum.

Agency Level II (moderate) — Unauthorized intentional or inadvertent release of information could seriously compromise effectiveness of Department or Agency. Could also include significant possibility of harm to individuals or serious violation of privacy. Normal protective requirements on multi-user systems for this level include those for Level I, plus access controls based on job function, and security anomaly detection as a minimum.

Agency Level III (high) — Unauthorized intentional or inadvertent release of information could gravely compromise effectiveness of Department or Agency. Could also include strong likelihood of grave harm or death to individuals. Normal protective requirements on multi-user systems for this level include those for Level II, plus strict compartmentation of types and levels of information, and stringent measures to protect information while in storage and in networks, all supported by a high degree of assurance.

Guidance on Use of Trusted Systems for Confidentiality Protection

The following concepts represent the core of NIST guidance on confidentiality protection via trusted systems. This guidance consists of a set of general principles applicable to all Federal agencies and computer systems that fall under the *Computer Security Act*. NIST is engaged in developing more comprehensive guidance on the use of trusted systems technology for confidentiality, integrity, and availability protection.

1. General Guidance — Risk Management Required. NIST recommends the use of trusted systems technology to agencies with significant requirements for adequate and cost-effective access control protection. Such requirements exist when there is a need for safeguarding principally the confidentiality and secondarily the integrity of information. In addition, the assurance process which is a part of trusted systems technology can help support system availability requirements. Cost-effectiveness is achieved when computer security controls, including trusted systems technology, are selected which are commensurate with the risk and magnitude of loss within a particular operating environment. This risk management process should balance security and performance requirements to provide for effective security and privacy of sensitive information in the system. Use of trusted systems technology, like any other security mechanisms, should substantially increase the protection when compared to acquisition, operating and maintenance costs of the security mechanisms.

2. Selection of Products from the EPL. Agencies with a need for systems with trusted technology features should select those systems from the National Security Agency's Evaluated Products List (EPL). If EPL products are not available, then agencies may select or design systems that best meet their security requirements using the TCSEC, the "Orange Book," as a guide [3].

3. Use of Class C2 Systems. Systems designed to meet C2 or higher classes of the TCSEC should first be considered when acquiring multi-user computer systems with a requirement to control user access to information according to need-to know and authorization. The C2 and other TCSEC criteria were designed to achieve confidentiality through improved access control. The same access control mechanisms can also be beneficial for helping to maintain information integrity.

4. Use of Division B Systems. Systems designed to meet the criteria of the B division of the TCSEC (especially B1 and B2) can be useful when acquiring multi-user computer systems with a requirement for mandatory separation of unclassified sensitive information and for which security labels can be established. Systems in that division are designed to enforce a mandatory access control or multi-level security policy. However, the cost benefit considerations discussed earlier are particularly important here.

A Potential Matrix for Confidentiality

In applying the guidance above to Federal unclassified systems which require confidentiality protection, the matrix in Table 1 is suggested for discussion purposes only. The matrix is based on the four levels of information sensitivity proposed above, and takes into account the different types or categories of sensitive information on a system. Access to types or categories of information is based principally on job-related need, while access to levels of that information is based on an estimation of trust.

It must be emphasized that determination of the requirement for a particular class of TCSEC trust (e.g., C2) must be based ultimately on cost effectiveness, as determined by a risk analysis, rather than on a simplistic matrix such as the following, which can serve only as a guide or rule of thumb.

Table 1 - Proposed NIST Guidance

Level of Disclosure Sensitivity	One Type and Level	2 or more Types or Levels
III	B1 (Note 1)	B2 - B3 (Note 2)
II	C2 (Note 3)	B1 - B2 (Note 4)
I	C1 - C2 (Note 5)	C2 - B1 (Note 6)
0	NR (Note 7)	NR (Note 7)

Notes to Table 1:

1. Organizations processing Agency Level III information (highly sensitive to disclosure) on multi-user systems should consider using systems designed to meet B1 as a minimum. The enforced labeling and reduced capability for propagation of user rights would significantly help protect this highly sensitive information.
2. Organizations processing more than one type of Agency Level III information on the same system should consider use of B2 minimum systems where the user populations desiring access to each type differ significantly or where there is a significant potential for harm from mis-identifying files or output products by type. This is also true when Agency Level III and lower levels of information of the same or different types are processed on the same systems simultaneously. When risk analysis shows there to be a need for better DAC, audit, trusted path, and assurance, the level of trust required could reach B3.
3. Organizations processing Agency Level II information (moderately sensitive to disclosure) on multi-user systems should consider using systems designed to meet C2 as a minimum.
4. Organizations processing more than one type of Agency Level II information on the same system should consider use of B1 minimum systems where the user populations desiring access to each type differ significantly or where there is a significant potential for harm from mis-identifying files or output products by type. This is also true when Agency Level II and lower levels of information of the same or different types are processed on the same systems simultaneously. When risk analysis shows there to be a need for better MAC, labeling, audit, and assurance, the level of trust required could reach B2.
5. Organizations processing a single type of Agency Level I information (minimally sensitive to disclosure) on multi-user systems should consider using systems designed to meet C1 as a minimum. When risk analysis shows there to be a need for better DAC, auditing, or object re-use control, the level of trust required could reach C2. Based on our observations, NIST believes most Federal department and agencies will require at least C2 for their multi-user systems, especially when information integrity requirements are considered.
6. Organizations processing more than one type of Agency Level I information on the same system should consider use of C2 minimum systems where the user populations desiring access to each type differ significantly. When risk analysis shows a significant potential for harm from mis-identifying files or output products by type, there may be a need for MAC, labeling, and better auditing. The level of trust required could reach B1.

7. No special disclosure protection would be required for systems which do not contain at least Agency Level I information.

DOD Approach[2]

This section presents a DOD approach to provide a method of "classifying" sensitive information and determining user trustworthiness. A proposal for minimum protection requirements (stated in terms of TCSEC classes) for confidentiality is also presented. DOD Directive 5200.28 [4], which implements the *Computer Security Act* for the DOD, defines sensitive unclassified information and states:

> ...sensitive unclassified information shall be safeguarded at all times while in AISs. Safeguards shall be applied so that such information is accessed only by authorized persons, is used only for its intended purpose, retains its content integrity, and is marked properly as required...

Position Sensitivity Levels

The OPM has broad oversight responsibility for the civilian personnel security program. The Federal Personnel Manual (FPM) [5] identifies personnel security as the process for complying with the national security interest requirements and discusses the need to determine personnel suitability as a requirement for Government employment with respect to a person's character, reputation, trustworthiness, and fitness as related to the efficiency of the organization.

OPM has established four position sensitivity levels and criteria for designating a given position at a particular level, as well as investigative requirements for each level. Satisfactory completion of the investigative requirements for a position sensitivity level may be used as a basis for determining the "trustworthiness" of an individual. The definitions of the four OPM position sensitivity levels follow:

NS - Non-Sensitive: Potential for impact involving duties of limited relation to the organization mission with program responsibilities which affect the efficiency of the organization. (National Agency Check and Inquiries)

NCS - Noncritical-Sensitive: Potential for moderate to serious impact involving duties of considerable importance to the organization mission with significant program responsibilities which affect the efficiency of the organization. (Limited Background Investigation)

CS - Critical-Sensitive: Potential for exceptionally grave impact involving duties of clearly major importance to the organization mission with major program responsibilities which affect the efficiency of the organization. (Background Investigation)

SS - Special-Sensitive: Potential for inestimable impact involving duties especially critical to the organization mission with broad scope and authority (e.g., overall direction of a major government program) or other extremely important responsibilities which affect the overall efficiency of the organization. (Special Background Investigation)

The FPM is applicable only to *civilian* positions, and there is no similar DOD guidance for military positions that require access to sensitive information. One approach for handling military positions requiring access to sensitive information is that, where appropriate, DOD components adopt the OPM guidelines for determining position sensitivity levels. Another approach is to establish a correspondence between the investigative requirements for the OPM position sensitivity levels and DOD clearances.

Information Sensitivity

For the second dimension of protection requirements, this approach provides a structure of non-hierarchical sets and three hierarchical levels of sensitive information. There are three steps in determining information sensitivity:

[2] This approach was derived from work supported by the NCSC under contract F19628-89-C-0001. The opinions expressed do not necessarily represent the position of any organization.

1. Determining non-hierarchical sets of information.
2. Determining the access control requirements for the information.
3. Determining the hierarchical information sensitivity level of the information.

Step 1 analyzes the information on a system to determine what (if any) non-hierarchical "sets of information" relating to specific subject areas exist. The term "sets of information" will be used to refer specifically to these non-hierarchical collections of specific information. Examples include PROCUREMENT SENSITIVE, PAYROLL, INVESTIGATIONS, MEDICAL, PERSONNEL, PROJECT XYZ, and GROUP ABC. A subject would require some specific type of approval for the set of information before being allowed access to it. This approval may consist of a formal authorization process (e.g., signing a non-disclosure form), or may simply be a matter of one's job function (e.g., all payroll clerks have access to payroll information). This approval process is typically outlined within an organization security policy. Sets of information are implemented to provide finer control over who has access to information within hierarchical sensitivity levels (even if the system only has a single hierarchical level). Non-hierarchical sets of information may also span multiple hierarchical levels. Access requirements might include demonstrated need-to-know for the performance of job-related functions, membership in a group, information ownership, or others.

Once the appropriate sets of information have been identified, step 2 involves determining what type of access controls are required. The TCSEC discusses two access control policies for trusted computer systems: Discretionary Access Control (DAC) and Mandatory Access Control (MAC). This approach specifically identifies two types of sets of information based on the access controls applied: information groups and categories. Information groups are defined to be less formal than categories, and may be more appropriate when the set of information is considered less sensitive, and therefore require less stringent technical controls. DAC may be used to protect information groups. Categories, on the other hand, are more formal and generally require a person have some formal access approval and/or security indoctrination before being allowed access. Because of their sensitivity, categories require stronger controls than DAC; MAC provides these additional controls. The appropriate official responsible for the system must make a determination as to the type of access controls required for each set of information. This determination is based on factors such as the sensitivity and number of the sets of information, the authorizations of users on the system, and the processing environment. By examining such factors, a decision is made as to how stringent the access controls must be for each set of information.

In addition to the non-hierarchical sets of information, step 3 of this approach defines three hierarchical levels of sensitive information (N1, N2, and N3). An unclassified level "U" as defined in the NIST approach (Level 0) may also be included here.

N1 - Low Sensitive Information: The unauthorized disclosure of N1 information would cause minimal identifiable damage to an organization mission or reputation or person.

N2 - Medium Sensitive Information: The unauthorized disclosure of N2 information would cause significant damage to a statutory responsibility of an organization. N2 information includes mission critical or organization operational information, and high technology related information which is restricted by law from exportation to certain countries. **N2 is the minimum recommended hierarchical information sensitivity level for both privacy information and proprietary information.**

N3 - High Sensitive Information: The unauthorized disclosure of N3 information would cause irreparable damage to an essential mission of an organization. Examples of N3 information are types of mission critical or organization operational information (defined to be higher in criticality than mission critical or organization operational information within the N2 category), and information that is life critical. The unauthorized disclosure of life critical information has the potential to result in the loss of human life.

Dollar impact ranges may be defined by each organization for the hierarchical information sensitivity levels. A risk analysis may help determine the appropriate dollar impact values for a system. In addition, the association of specific civil or criminal penalties with the unauthorized disclosure of the information may be a driving force in determining the appropriate sensitivity level.

Protection Requirements

Table 2 presents a matrix with the suggested minimum protection requirements for confidentiality. The computer security requirements recommended are minimum values. Environmental characteristics must also be examined to determine whether a higher class is warranted. Factors that might argue for a higher evaluation class include the following:

1. High volume of information at the maximum information sensitivity level.
2. Large number of users with low position sensitivity levels.
3. Specific civil/criminal penalties associated with the unauthorized disclosure of the information.

Table 2 - Proposed DOD Guidance

		Maximum Information Sensitivity						
		U	N1 or N1 Groups	N1 Catgs	N2 or N2 Groups	N2 Catgs	N3 or N3 Groups	N3 Catgs
Minimum Position Sensitivity Level	U	C1	B1	B1	B1	B1	B2	B2
	NS	C1	C2	B1	B1	B1	B1	B1
	NCS	C1	C2	B1	C2	B1	B1	B1
	CS	C1	C2	B1	C2	B1	C2	B1
	SS	C1	C2	B1	C2	B1	C2	B1

Notes to Table 2:

Although there is no recommended minimum for dedicated mode systems, the integrity and denial of service requirements of many systems warrant at least class C1 protection.

Class C2 is the minimum recommendation for system high mode.

Class B1 is the minimum recommendation whenever categories have been identified.

The minimum recommended level of trust for environments processing sensitive information is Class C2. This is based on DOD Directive 5200.28. The C2 level provides DAC, which controls access to information based on permissions granted to the user, but does not support internal labeling of information. In addition, C2 provides individual accountability and the maintenance of audit trails.

B1 is the minimum recommendation whenever categories have been identified. In addition, even within a system high environment, B1 may be appropriate if specific civil/criminal penalties can be imposed for the unauthorized disclosure of the information, or if all the information is considered critical-sensitive. The primary reason for the B1 minimum recommendation for these environments is that B1 is the first TCSEC class to provide MAC and labeling. The combination of both MAC and DAC provide for a finer granularity of access control. MAC also prevents the free passing of access privileges, which is important in those environments with higher levels of information sensitivity, and a greater disparity between the minimum position sensitivity level and the maximum information sensitivity level. MAC is also recommended whenever information at two or more hierarchical levels is being processed, even if everyone is fully authorized.

Canadian Approach

This section presents an approach currently proposed in Canada for determining the minimum computer security requirements for the protection of designated (sensitive) information. This approach is based on three factors: (1) a minimum user screening level, (2) maximum data sensitivity, and (3) the operating mode of the system. Based on these three factors, the approach provides a guideline for the selection of a Trusted Computing Base (TCB) for a particular application or environment. The TCB requirements are stated in terms of classes from the DOD TCSEC [3].

Minimum User Screening Levels

Appendix F (Personnel Screening Standards) of the Security Policy of the Government of Canada [6] provides standards for the personnel screening process for individuals to be employed by the Government of Canada. These standards apply to all personnel employed either directly or indirectly (e.g. contracted services) by the Government. Personnel screening is carried out according to the highest level of information and assets which will be accessed in the normal performance of assigned job duties or contract requirements. For access to sensitive information, personnel screening involves the assessment of a person's reliability. There are two types of reliability checks: (1) a Basic Reliability Check and (2) an Enhanced Reliability Check.

Basic Reliability Check (BRC): a condition of employment to the Public Service of Canada for all individuals who are appointed or assigned to a position in the Public Service or who are under contract, for more than six months and who will have regular access to government premises. It involves a declaration, that is included in an individual's consent to screening, concerning any conviction for a criminal offence for which a pardon has not been granted; verification of personal data, educational, professional or trade qualifications, and employment data and references; and an optional criminal records name check.

Enhanced Reliability Check (ERC): required when the duties of a position, or contract requirements, demand a significant degree of access to designated (the Canadian term for sensitive) information or assets. Factors which are considered when determining the significance of access include the sensitivity, value, or volume of information or assets and the frequency of access. An enhanced reliability check involves a basic reliability check, a fingerprint check, and a credit check.

Data Sensitivity

Appendix C (Security Organisation and Administration Standards) of the Security Policy [6] provides operational standards for the organisation and administration of the security of classified or designated information and assets. Government institutions control information that lies outside the national interest category and, therefore, may not be classified. It may nevertheless be sensitive, merit designation as such and require enhanced protection. Such information is generally identified in the *Access to Information Act* and the *Privacy Act*. However, not all designated information is of the same nature. Some is "particularly sensitive," the compromise or unauthorized disclosure of which could cause serious or extremely serious injury. Examples could include medical records or details about confidential police sources on organized crime. Institutions in the government are required to conduct a thorough review of information holdings and assets, and to identify material that requires designation as sensitive material.

Each institution must develop a classification guide. All information and assets which have been determined to have sensitivity in other than the national interest are to be marked PROTECTED. This is the standard marking which signals the application of minimum standards. Institutions have an option of adding the letter A, B, or C to the marking PROTECTED to indicate the need for varying degrees of security measures. The letter A can be added to the marking to indicate the requirement for minimum protection standards resulting in the marking PROTECTED A.

Institutions are required to identify particularly sensitive information and apply security measures based on a threat and risk assessment. To counter additional threats that may apply, more stringent security measures are recommended for the protection of designated information that is particularly sensitive. Institutions have the option of adding the letter B to the marking PROTECTED to signal the need for additional security measures. Because of the varying nature of particularly sensitive designated information and related threats, it is not to be assumed that the application of safeguards will be the same from one institution to the next.

In a few cases, institutions hold designated information that, if compromised, may cause extremely serious injury, such as loss of life or serious financial loss. In such cases, special security measures may be warranted and institutions have the option of adding the letter C to the marking PROTECTED to signal the need for special stringent safeguards.

In all cases of extended markings (i.e. A, B, or C), it can not be assumed that the application of safeguards from one institution to the next will be the same. The policy therefore recommends a written agreement between the security offices of the institutions involved in sharing such information.

Operating Modes of a System

There are three operating modes which are considered in determining the protection requirements for systems processing sensitive information:

Dedicated mode where all users associated with the system have a valid clearance, approved access, and a valid need-to-know for ALL information on the system.

System high mode where all users associated with the system have a valid clearance and approved access for ALL information on the system, but do not have a valid need-to-know for all of the information on the system.

Multi-level mode where all users associated with the system do not have a valid clearance and approved access for all information on the system, and have a valid need-to-know for SOME of the information on the system.

Determining TCB Requirements for Government of Canada Computers

As stated earlier, the guideline for determining the required TCB is based on the operating mode, the data sensitivity, and the user clearance. These three values are used to find an element in the following two tables. The procedure used is as follows:

(1) Find the entry in Table 3 corresponding to the minimum user screening of any user associated with a processor or system and the maximum data sensitivity on the processor or system.

(2) If the entry in Table 3 is D/SH (i.e. Dedicated or System High), then Table 4 is referenced to find the appropriate TCB level corresponding to the operating mode of the system.

Table 3 - Proposed Canadian Guidance

	Maximum Data Sensitivity		
Minimum Screening	U	NATO Restricted	Protected (A/B/C)
U	C2	B1	B2 (Note 2)
BRC	D/SH	D/SH	B2 (Note 2)
ERC	D/SH	D/SH	D/SH

D/SH means that this instance is covered by Table 4

Notes to Table 3:

1. All TCB requirements may be reduced by one level on the basis of a threat/risk assessment when operating in a closed environment or when user access is restricted to limited function/menu-driven terminals.
2. If no "particularly sensitive" information is involved (PROTECTED A), a B1 TCB is acceptable. However, some "particularly sensitive" information (PROTECTED C) may warrant a B3 TCB based upon a threat/risk assessment.

Table 4 - Proposed Canadian Guidance

	Maximum Data Sensitivity	
Operating System Mode	Unclassified	Designated (NATO Restricted or Protected)
Dedicated	D (Note 1)	C1 (Notes 2/3)
System High	C1	B1 (Note 4)
Multi-level	C2 (Note 5)	see Table 3

Notes to Table 4:

1. The security policy of the Government of Canada requires that even unclassified/undesignated information be afforded " ...basic protection reflecting good management practices." Therefore unclassified/undesignated information may require some protection, such as that provided by a C1 TCB when operating in dedicated mode with more than one user.
2. A C2 TCB is recommended when processing "particularly sensitive" designated information (PROTECTED B and C)
3. For dedicated mode, DAC is not required. However, object reuse (OR), identification and authentication (I&A), and audit (AUD) features are required when processing "particularly sensitive" designated information (PROTECTED C). These requirements can be satisfied using subsystem components having OR/D2, I&A/D2 and AUD/D2 ratings as defined in [7].
4. A B1 TCB is recommended when "particularly sensitive" information is processed with other designated data (when two or more of PROTECTED A, B and C are processed concurrently) to avoid the necessity of manual downgrading of less sensitive output. Mandatory Access Control is not required in System High Mode of operation so this feature of B1 TCBs may be disabled.
5. All Government employees require a BRC as a condition of employment. Access to unclassified/designated Government information by unscreened individuals (those lacking a BRC) constitutes a form of multi-level operation. A C2 TCB is recommended for Identification and Authentication and for Audit capabilities.

COMPARISON OF APPROACHES

The previous section outlined three proposed approaches for the protection of sensitive information against unauthorized disclosure. The *Computer Security Act of 1987* is the primary legal basis for the protection of sensitive information in the U.S. The NIST approach provides basic guidance for protection via the implementation of trusted systems technology, and the DOD approach presents a proposed approach for the U.S. Defense community. For comparison purposes, Canada's proposed approach is also included. There are strong similarities between the three approaches; however there are some interesting differences. These are discussed below.

The lack of any well-defined (or even partially accepted) standards for user trustworthiness results in the greatest variation among the approaches. The NIST guidance in this area must be very broad and general due to the vast differences among Federal Agency missions and objectives. User trustworthiness may be stated as a quantifiable metric (such as using clearance levels based on well-defined background investigations) or a "warm fuzzy" metric (such as job-related need-to-know, years of service, demonstrated prior trustworthiness). The DOD approach adopts the OPM 5 position sensitivity levels (uncleared, NS, NCS, CS, and SS) for which background investigation requirements are defined. However, the determination as to what OPM level is required for accessing various levels of sensitive information is not straightforward. The Canadian personnel screening requirements is more simplified in that only 3 levels of investigation (uncleared, BRC, and ERC) are defined. The Canadian approach also provides a mapping between the reliability checks to designated levels of sensitive information.

The data sensitivity dimension is similar in all three approaches; however, the emphasis in each differs. Both the NIST and DOD approaches define three hierarchical levels of sensitivity. The Canadian levels Protected A/B/C are not hierarchical in that access to Protected C does not provide access to Protected B, etc. In this sense, A/B/C are very similar to the "sets of information" described in the

DOD approach. The NIST approach also discusses non-hierarchical "types" (i.e., categories) of information and factors the types of information into the determination of the TCB requirements. The DOD approach emphasizes that the sets of information (non-hierarchical) are necessary to provide a finer grain of access control within hierarchical sensitivity levels. However, even within the structure defined by each approach, the problems of sharing information across security domains still exist unless some labeling standards are adopted throughout the communities of interest.

Another area in which the three approaches differ in emphasis is in their consideration of other factors for determining the appropriate TCB level. The Canadian approach explicitly uses operating mode in its calculation of protection requirements, and within the footnotes to the tables, references open/closed environment and user access mode. The DOD approach also discusses operating mode in the footnotes to the protection requirements matrix. Consideration of such factors is implicit in the NIST approach. NIST recommends that the determination of adequate protection requirements be based upon an analysis of the security risks of the environment.

Finally, in the area of TCB level recommendations, the three approaches again are very similar. In all three approaches, C2 is the minimum recommendation for any environment (except for dedicated mode). Both the NIST and DOD approaches recommend C2 for system high mode processing. The Canadian approach recommends B1 for system high when "particularly sensitive" information is processed with other designated data (e.g., when two or more of Protected A, B, and C are processed concurrently). B1 is the minimum recommendation for multilevel mode in the DOD approach. The NIST approach also *suggests* B1, although it allows more flexibility in choosing C2 or B1 by relying on risk analysis results to determine the minimum security requirements. The Canadian approach recommends a B2 TCB for multilevel mode, although a B1 is acceptable if only Protected A is involved. However, the Canadian approach also states that a B3 TCB may be warranted for Protected C information based upon a threat/risk assessment. The DOD and NIST approaches also recommend B2 when high-sensitive (Agency Level III) information is being processed.

In summary, all three approaches stress that computer systems that process sensitive information require minimum computer security requirements. Many of the tools and mechanisms developed for handling classified information within computer systems also apply to computer systems that process sensitive information. The authors hope that work and progress will continue in the area of sensitive information protection and that a framework will emerge for all communities of interest.

ACKNOWLEDGMENT

Special thanks is given to Dr. Marshall Abrams of MITRE for initiating this cooperative effort.

REFERENCES

1. Public Law 100-235, 101 STAT. 1724, *Computer Security Act of 1987*, 8 January 1988.
2. Office of Management and Budget, *Management of Federal Information Resources*, OMB Circular No. A-130, 12 December 1985.
3. National Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD, December 1985.
4. Department of Defense, *Security Requirements for Automated Information Systems (AISs)*, DOD Directive 5200.28, 21 March 1988.
5. Office of Personnel Management, *Federal Personnel Manual*, 6 January 1984.
6. Treasury Board of Canada, Secretariat, *Security Policy and Standards*, December 1989.
7. National Computer Security Center, *Computer Security Subsystem Interpretation*, NCSC-TG-009, Version 1, 16 September 1988.

Considerations for VSLAN® Integrators and DAAs

Greg King
Verdix Corporation
14130-A Sullyfield Circle
Chantilly VA 22021
703-378-7600
GKing@DOCKMASTER.ARPA, king@verdix.com

ABSTRACT

The Verdix Secure Local Area Network (VSLAN) is a Network Trusted Computing Base (NTCB) designed to interconnect host systems, workstations, printers, routers, gateways, terminals, and other devices operating at differing security modes and accreditation ranges. As of this writing, the product is in formal evaluation as a B2 MDIA network component. It is intended that system integrators will use the VSLAN as a NTCB foundation for trusted local area networks. In this paper, based on our integration experiences thus far, we identify technical considerations for future integrators, describe one of our integration experiences, and discuss some of the relevant implications for Designated Approving Authorities (DAAs). A significant implication for DAAs, as well as integrators, is the fact that fiscal pressures, the assurance ranges of available TCSEC TCBs, and the lack of evaluated network protocols and applications may result in integrations that are secure but rarely composite B2 networks consistent with the Single Trusted System (STS) view of the TNI. We fully believe that the resulting integrations will be accredited to process classified information but provide evidence that many factors will combine to require DAAs to adopt Interconnected Accredited AIS (IAA) or hybrid IAA/STS views of the resulting trusted LANs.

1. Introduction

As background, we begin by providing a brief technical description of the VSLAN, by describing the two TNI network views, and by describing the current results of the NCSC commercial product evaluation.

1.1. VSLAN Technical Summary

The VSLAN consists of multiple trusted network interfaces, referred to as Network Security Devices (NSDs), and a dedicated central management facility known as the Network Security Center (NSC). The architecture is shown in Figure 1.1. Provided TNI Part I security services are: mandatory access control (MAC), discretionary access control (DAC), auditing, and identification and authentication (I&A). Provided TNI Part II security services are: communications field integrity, continuity of operations, protocol based denial of service protections, network management, and data confidentiality. In addition to these security services the NSDs provide IEEE 802.3 media access[1]. As further background, it is beneficial to provide the relevant VSLAN definitions of MAC, DAC, audit, and I&A.

For MAC, subjects are host or workstations and objects are datagrams. For each transmit and receive operation, NSD's perform MAC checks that insure the MAC label attached to each datagram is within the accreditation range of the transmitting or receiving NSD[2]. The Network Security Officer (NSO) defines the accreditation range of each host. In support of the MAC service, these accreditation ranges are downloaded from the NSC to each NSD upon its initialization.

For DAC, the NSO, and only the NSO, is provided the ability to authorize or revoke associations (i.e. communications paths) between a principal and any NSD. These associations are two-way; both transmit and receive. A principal is the specific individual responsible for operation of a NSD.

For audit, audit events are security relevant activities (e.g. key distributions, policy violations, etc.), security officer operations, and status changes. While it is true that objects are datagrams and subjects are hosts, the TNI requirements for introduction of objects into a user's address space are balanced with performance desires by allowing selection of this type of audit but not requiring it. Audit data is generated by NSDs and the NSC and stored and protected on the NSC. Audit data is never lost.

For I&A, principals are provided with an authentication token known as a Datakey. The Datakey is an EEPROM device that authorizes an individual to use one and only one NSD. NSD devices contain a keyceptacle device for insertion of the key. Without the correct key, network access is denied. A related feature of this mechanism is a principal identifier that is bound to the Datakey. This principal identifier is exported with each datagram.

1.2. TNI Network Views

This paper frequently references the two possible TNI network views. For convenience, we review the definitions of those views. The first view is referred to as the Single Trusted System (STS) view. Characteristics of the STS view are: a single coherent security architecture, a common level of trust throughout the system, and a single accrediting authority. The second view is referred to as the Interconnected Accredited AIS (IAA) view. Characteristics of the IAA view are complex, heterogeneous, combinations that lack a uniform level of trust

[1] LLC (layer 2) and layer 3-7 protocol S/W is presumed to reside in the host or workstation.

[2] On transmit operations, hosts or workstations are responsible for attaching the correct MAC label. A host to front end (HFE) protocol is defined for this purpose.

and often include multiple accrediting authorities. STS evaluations result in a network class (e.g. A1, B3, B2) while IAA views provide guidance on appropriate interconnection strategies.

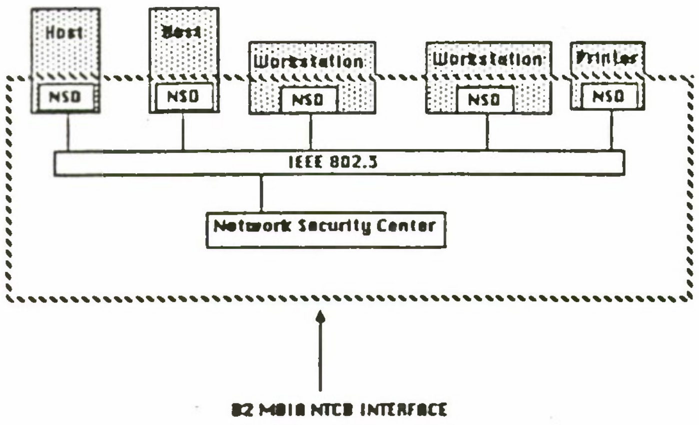


Figure 1.1. VSLAN Architecture.

While it is preferable that a certification use the STS view (because of the conveyed hierarchical trust structure) it is important to note both that: a). in heterogeneous network architectures that include hosts of differing trust levels the resultant rating may not be a representative trust rating for all network connections and b). the STS view and the IAA view are not mutually exclusive.

An example of the misrepresentation that can occur with the STS view is a network that consists of two A1 systems and two B2 systems. In this example, adopting the STS view would result in a B2 rating. The resultant B2 rating might not be representative of the trust that could be placed in the A1 systems and the network connections between them.[3] As noted, the STS view and the IAA view are not necessarily mutually exclusive. This means a network of IAA's can consist of one or more separately accreditated STSs.

1.3. NCSC Commercial Product Evaluation

As of this writing, the VSLAN is in the final stages of a commercial product evaluation at the NCSC. Product evaluation began in 1986 and is currently scheduled for completion in the middle of 1990. The evaluation is being conducted with respect to Version 1 of the TNI. So far, the evaluation results conclude that the VSLAN is a B2 MDIA network component providing the previously mentioned security services.

A significant characteristic of the evaluation is the STS orientation that acknowledges the product is not a complete network but assumes that integrators and end users will ultimately use the VSLAN as the basis for STS view B2 networks. The practical aspects suggest this may not be a good assumption. The assumption, however, is rooted in the TNI's TCSEC origins.

An additional characteristic of the evaluation that is often neglected is the evaluation of the VSLAN with respect to the nine security services (TNI Part II) that are claimed to differentiate network and standalone environments. As of this writing, the lack of an objective scientific method for applying the TNI Part II evaluation results to a specific environment diminishes their usefulness.

2. Organization

The remainder of this paper is organized into four additional sections. Section 3 introduces the fundamental underlying architectural assumption for the examples and discussions that follow. That assumption is that the VSLAN forms a NTCB foundation to which integrators will add supplemental NTCB mechanisms to form trusted local area networks. Section 4 introduces and describes necessary supplemental NTCB mechanisms. Section 5 describes one of our integration experiences thus far. Section 6 discusses accreditation issues related to the IAA network view. Finally, section 7 provides conclusions.

3. Network Architectures

Any trusted local area network based on the VSLAN will likely include a network security policy enforced by a well defined network trusted computing base (NTCB). As defined in the TNI, a NTCB, is the totality of protection mechanisms with a network system -- including hardware, firmware, and software. While the VSLAN component can be expected to be a significant portion of that NTCB, additional NTCB mechanisms will also be required. These additional NTCB mechanisms may include trusted operating systems, trusted device drivers, trusted subsystems, trusted communications protocols, and possibly additional security protocols (e.g. SP2, SP3, SP4). Figure 3.1 shows an example NTCB composed of the VSLAN, trusted device drivers, trusted communications protocols, and individually evaluated TCSEC TCBs (i.e. trusted operating systems) of varying assurances.

[3]TNI, Version 1, Appendix C, pg. 245.

A significant implication of the chosen architecture relates to the composite assurance rating of the NTCB. When the STS view is assumed for the entire NTCB, the composite NTCB assurance rating has an upper bound equal to the lowest TCSEC TCB assurance rating of any LAN host. For example, assuming, an integration of two C2 hosts and two A1 hosts using the VSLAN, and assuming that the VSLAN is configured to prevent the A1 and C2 hosts from communicating (with a B2 level of assurance) the composite assurance rating is, nevertheless, C2.

As subsequent sections will suggest, even with the B2 VSLAN component, and sufficient numbers of B2 TCSEC TCBs, building integrated NTCBs will still require supplemental NTCB mechanisms that may not be subjected to NCSC commercial product evaluations. Consequently, particularly in the short term, despite the fact that the result integrations may be secure and may adequately counter threats, DAAs may find it difficult to view the resulting integrations using STS views that yield particular composite assurance ratings.

4. Supplemental NTCB Mechanisms

In this section, we describe some of the necessary supplemental NTCB mechanisms when the VSLAN is used as an NTCB foundation for trusted local area networks. We address implications for DAAs and integrators concerning the integration of trusted operating systems, communications protocols, network applications, and the necessary device drivers.

4.1. TCSEC COTS TCBs

In light of the assurance issues as related to STS views of the resulting integrations, it is informative to roughly quantify the availability of TCSEC evaluated COTS TCBs to determine the likelihood that the B2 VSLAN component will be used to construct STS view B2 trusted LANs. As Table 4-1 indicates, currently there is a large cluster of COTS TCBs at the C2 assurance class. Potentially, in the relative short term, there appears to be an increasing cluster of Unix or Unix like TCBs at the B1 class.

In the short term, this distribution will greatly influence the resulting VSLAN integrations. Based on the Table 4-1 data, we assume driving short term forces to incorporate many C2 and B1 hosts. This is not to state that we believe that the VSLAN will always be used to build less than B2 NTCBs but rather to suggest a.) that few B2 TCSEC are available for integrators and b.) that the resulting short term integrations will likely contain both B1 and C2 hosts and therefore exhibit different levels of trust at NTCB interfaces. The resulting integrations may undoubtedly be secure but rarely the type of B2 STS view network envisioned by the commercial product evaluation or the TNI, Part I. Despite the B2 NTCB foundation provided by the VSLAN, DAAs may be forced to adopt IAA or hybrid IAA/STS views of the resulting integrations.

In the longer term, an evolution towards the type of B2 STS view network envisioned by the TNI Part I is possible as significant B2 TCSEC TCBs become available for inclusion in the NTCB. Two significant Unix TCBs appear waiting. Both are in evaluation at the B2 class. However, even when additional B2 TCSEC TCBs become available, an additional obstacle may have to be overcome before B2 STS view trusted LANs can be developed. That obstacle is the evaluation of communications protocols provided by the B2 TCSEC TCBs. If these protocols are part of the TCB, they must be subjected to the same rigorous assurance processes imposed on other parts of the TCB. At class B2, this implies significant software engineering methods and testing which may delay commercial product evaluations or result in B2 TCSEC evaluations that exclude networking applications or protocols from the TCSEC evaluation.

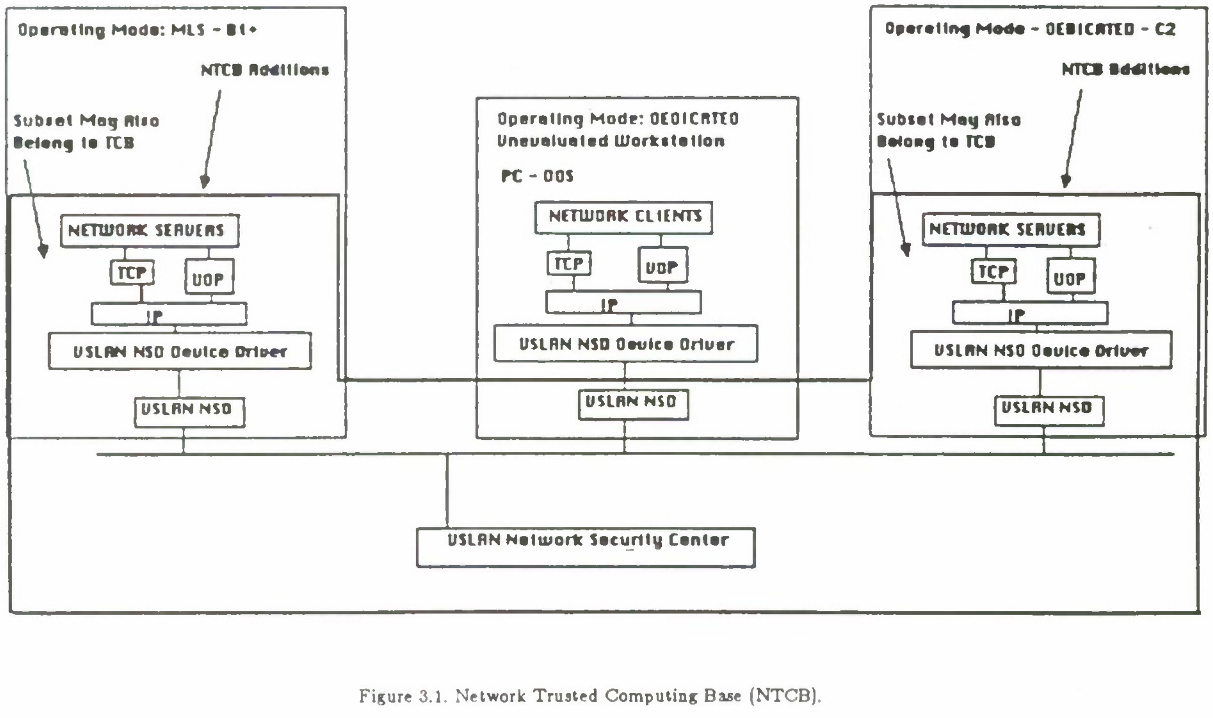


Figure 3.1. Network Trusted Computing Base (NTCB).

Class	Actual	Potential
A1	1	2
B3	0	0
B2	1	3
B1	2	8
C2	10	10+

Table 4-1. EPL Entries[4].

4.2. Communications Protocols and Network Applications

In the previous section, we suggested that we believe, in the short term, VSLAN integrators will be more likely to integrate C2 and B1 hosts than B2 hosts. In this section, we suggest that the communications protocols and network applications tbat are part of the supplementary NTCB mechanisms and often part of the TCSEC TCB kernels may consist of largely unevaluated software. This is not to suggest that we believe that unevaluated protocols and applications themselves are inherently flawed or that they don't address security threats, or tbat tbey haven't been adequately tested but simply that DAAs may not be able to claim a specific TCSEC or TNI class of assurance for a given implementation. An additional complicating theoretical but less practical concern is that if the protocols themselves are configured to be part of the TCSEC TCB kernel[5] they may invalidate the original TCSEC TCB rating. This is an obvious issue that has implications for DAAs and the STS view of any VSLAN based trusted LAN.

Despite the fact that we are suggesting that most supplementary communications protocols and network applications are likely to be unevaluated we believe that it is these very protocols and applications that integrators and DAAs must examine and evaluate most closely. They must be examined and judged according to their ability to counter specific security threats. It seems most likely that a pragmatic approach that ensures adequate countering of threat as opposed to approaches that ensure a specific level of TCSEC or TNI assurance will develop. At this point, we turn to a discussion of the some of the threats that must be considered when adding communications protocols to the VSLAN NTCB.

4.2.1. Threats

Often, one of tbe most cited and well known security threats to a local area network is wiretapping; both passive and active. For VSLAN integrators and DAAs wiretapping is likely to be the least serious threat. The VSLAN counters the wiretapping threat by providing DES encryption and requiring Protected Wireline Distribution Systems (PWDS) when greater than unclassified but sensitive information is processed by the network. Additionally, VSLAN NSDs receive only traffic addressed to them and traffic for which they are authorized. That is to say, VSLAN NSDs can not be configured to operate in a promiscuous mode and all packet transmissions and receptions are mediated according to the NSD's accreditation range and the NSD's discretionary access control list.

We anticipate that the more important threats that must countered and addressed will originate from authorized users taking advantage of security holes introduced by particular implementations of the communications protocols and network applications themselves. These types of threats have been well documented in other sources. Concerning the Berkeley Unix implementation of the TCP/IP protocols and the associated Berkeley Unix applications, one of the most recent and revealing sources is [Bell89].

While it is true that the risk of the following threats can be reduced to some extent by appropriately configuring the VSLAN DAC lists, we are concerned less with those instances where unauthorized users have been prevented access by VSLAN MAC and DAC features than with the more interesting case where an authorized user is attempting to abuse the VSLAN MAC and DAC privileges given him by the NSO.

One of the more significant threats for a Berkeley Unix implementation is the ability to establish a TCP connection by predicting initial sequence numbers. By predicting initial sequence numbers, an authorized user would be capable of impersonating a trusted host, establishing a connection, and possibly causing remote command execution. Fortunately, it is possible to counter this threat to some extent by comparing non circumventable tamperproof VSLAN NTCB source addressing information with IP source addresses on packet reception. Otber methods for countering the threat include elimination of network applications that don't provide sufficiently strong authentication measures.

Even when network applications include identification and authentication mechanisms, integrators and DAAs must take care to insure that individual connections are authenticated. For example, some network applications (e.g. FTP) require multiple control and data connections. Authentication and control information is exchanged over the control connection and user data is exchanged over the data connections. Following successful autbentication, data connections are established. Unless the client verifies the port number before establishing the data connection, data can be received from a malicious source. Details of this type of threat are well described in [Tsai89]. Countering this threat will generally involve requiring the network application to receive and verify port numbers for subsequent data connections.

Other threats include tbe ability to subvert the boot process by abusing remote booting mechanisms that make use of reverse ARP and TFTP protocols. Nearly all protocols and applications will introduce additional threats. Thorough examination of each threat is beyond the scope of this paper. Some implementations will counter or eliminate the threats, but may do so at the cost of interoperability (e.g. requiring port numbers before accepting data connections).

4.3. Device Drivers

Operating system device drivers for the VSLAN NSD are a required supplemental NTCB mechanism. The operating system device driver controls tbe VSLAN NSD by managing a 64K bank of dual port RAM and by participating in a Host-to-Front-End (HFE) protocol with the NSD. Specific security issues that integrators and DAAs should consider include: non-circumventability, security labeling, source address authentication, and network to IEEE 802.3 address resolution.

[4]Information is from NCSC. Taken from actual EPL entries, published potential EPL entries, and published product bulletins.

[5]As it must be for most Unix implementations.

4.3.1. Non-Circumventability

Because VSLAN mediation depends on correct host inputs (e.g. security labels) integrators and DAAs must insure the non circumventability of VSLAN device drivers. That is, untrusted user programs must be unable to control the VSLAN device. All access must be accomplished exclusively through the VSLAN device driver that resides in the operating system kernel.

Insuring the non-circumventability and tamperproof nature of the device driver may require restricting access to character special files that control physical and kernel virtual memory (e.g. /dev/mem, /dev/kmem).

4.3.2. Security Labeling

As earlier noted, VSLAN NSDs accept MAC security labels from their associated operating system device drivers through a HFE protocol. As the device driver participates in the HFE protocol, it unambiguously binds a MAC security label to each datagram as it requests transmission. Subsequently, the NSD compares the MAC security label with the current accreditation range to determine whether the requested datagram transmission should be accepted or rejected.

For single-level NSDs and their associated hosts, the security labeling performed by the device driver is straightforward. All datagrams presented to the NSD receive the same security label. The security label provided by the device driver is static. If desirable, device drivers can read the appropriate security label (i.e. accreditation range) from the NSD.

For multi-level NSDs and their associated hosts, the security labeling performed by the device driver may be more complex. Specifically, the device driver must dynamically determine the appropriate security label based on external inputs. These external inputs may include message type for control datagrams (e.g. ICMP, GGP, RIP, ARP) not originating in user processes, and user process labels for datagrams originating in user invoked network applications. For example, the correct security label of ARP and ICMP replies will be the security label of the incoming ARP or ICMP message. The security label for these messages is unrelated to the security labels of any open connection or user process.

In the most straightforward scenario, much of the burden associated with determining the correct security label will be handled by the communications protocols residing above the VSLAN NSD. Specifically, some implementations of upper layer protocols may support the communication of security labels through the use of a new type of IP security option known as the Commercial IP Security Option (CIPSO). In these instances, device drivers for multi-level NSDs could retrieve the correct security label for all datagrams (except ARP) from the security option.

For upper layer protocol implementations that do not support the CIPSO (or an equivalent), it seems likely that integrators will need to modify the input and output routines of the associated protocols to allow the appropriate security labels to be communicated. Description of the appropriate modifications is beyond the scope of this paper. However, at a certain level of abstraction, such modifications ultimately provide the same functionality as CIPSO implementations (although they may not interoperate).

Fundamentally, as the reader will have noted, whether CIPSO or otherwise, multilevel NSDs will require that the upper layer communications process security labels. As of this writing, we know of only one COTS TCP/IP package that is being modified to support the CIPSO. The package, as well as the modifications, are unevaluated software. The DAA implications for an STS view of the resulting integration are obvious.

4.3.3. Source Address Authentication

Some of the threats discussed in Section 4.2.1 involved the impersonation of network hosts. That section suggested that the resulting threats could be countered, to some extent, by requiring the NSD device driver to compare IP source addressing information with VSLAN NTCB source addressing information.

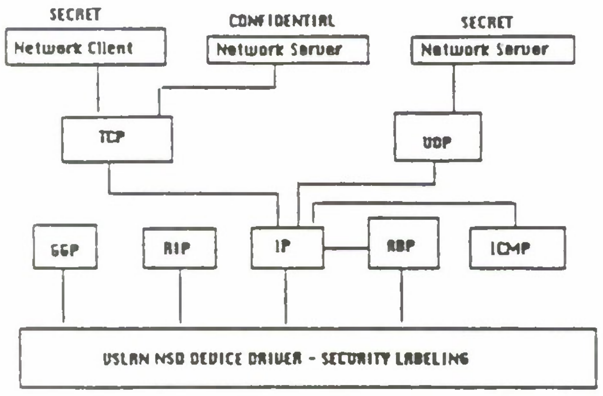


Figure 4.3.2. Classification Depends on Message Type or Label of User Process.

4.3.4. Network to IEEE 802.3 Address Resolution

An additional consideration for integrators and developers of NSD device drivers is the address conversion support to be provided (e.g. ARP) when converting IP addresses to IEEE 802.3 LAN station addresses. The underlying VSLAN issues that require consideration are: a.) the VSLAN's inability to support direct multicast or broadcast addressing, and b.) the need to correctly label layer 2 protocol messages such as ARP without CIPSO or equivalent labeling support[6].

Integrators must consider the fact that standard ARP implementations generate ARP requests to the IEEE 802.3 broadcast address. As the VSLAN component does not support broadcast or multicast addressing, these broadcast requests must be converted into multiple point to point transmissions or alternate schemes for address resolution must be adopted.

Recent integrations have used both solutions. To convert broadcasts to multiple point to point transmissions, the VSLAN's device driver watches for the IEEE 802.3 broadcast address (all ones) duplicating the transmission for each element of the NSD's discretionary access control list.

In other integrations, ARP has been eliminated and replaced by an addressing scheme that assumes that the last octet of the internet address for a node is the same as its VSLAN NTCB identification number (i.e. its NSD ID).

Yet other integrations have adopted a hybrid approach, where some nodes generate ARP requests, other nodes use the last octet of the internet address, and a single ARP server exists to respond to ARP replies.

In general, integrators should note that ARP will almost certainly require elimination from multilevel VSLAN nodes because of the inherently difficult problem presented by the requirement to correctly label ARP requests[7]. This is based on the assumption that an MLS node will convert an ARP request into multiple point to point transmissions and that all single level nodes with which it communicates are not at the same level and category set as the converted ARP request. Because of the MLS node's choice of label, some of the receiving nodes may not be at the same level and category set resulting in VSLAN MAC audits.[8] See Figure 4.3.4

Integrators should also note that the absence of broadcast support and the fact that an MLS nodes' conversion of broadcasts to multiple point to point transmission will cause MAC violations have broader implications. For example, identical issues exist when considering applications that utilize broadcast techniques (e.g. rwho, yellow pages). Unmodified rwho and yellow pages applications can be run on VSLAN nodes but their broadcast usage probably makes them better suited to single level, single category VSLAN nodes. When run on these nodes, the broadcast requests are easily converted to correctly labeled, multiple, point to point transmissions.

5. Integration Experiences

In this section we describe one of our recent integration experiences. Strictly speaking, because we were required to include unevaluated software in our integrated NTCB, we are unable to claim that our integration meets all of the B1 assurance requirements (i.e. we have not subjected the integrated NTCB to an NCSC commercial product evaluation, we have not prepared a TFM for the integrated NTCB, etc.) but we certainly believe that the integrations provides all of the B1 security features and most importantly we believe the integration adequately counters known threats.

5.1. An Integrated Trusted LAN

As of this writing, in our lab, we demonstrate a sample trusted LAN that uses the VSLAN NTCB as a foundation. Hardware includes an AT&T 3B2 minicomputer with a VSLAN NSD, numerous IBM PC ATs, each containing VSLAN NSDs, and a VSLAN NSC. As of this writing, we provide virtual terminal services for the PCs. We use five basic security assertions to describe the security aspects of our

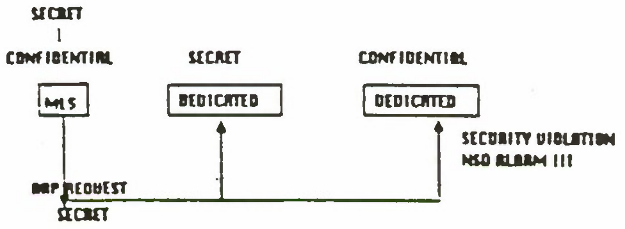


Figure 4.3.4. Multi-level VSLAN Node Attempts ARP Request.

[6] CIPSO or equivalent labels were useful for labeling protocol messages originating at or above layer 3 but such labels are unavailable for protocol messages originating below layer 3. In this instance, because ARP resides at the boundary between layer 3 and layer 2, CIPSO labels are unavailable.

[7]ARP is generally a protocol that doesn't easily port to the VSLAN because security labeling and the protocol's broadcast nature are in conflict. Another motivation for eliminating ARP is the desire to eliminate the potential denial of service attack that can be mounted by continually requesting a connection to a non-existent address.

[8]One of these nodes may be the legitimate node responsible for generating the ARP reply.

integration. Those security assertions are as follows:

SA1: Authorization

A user can conduct a remote terminal session with the AT&T System V MLS minicomputer only if the user's userID and password appears in the AT&T 3B2 /etc/passwd and /mls/passwd files and the user's userID appears in the VSLAN's NSC principal database. Additionally, the maximum classification of tbe user's userID must dominate the requested session classification.

SA2: Classification

All remote terminal sessions are conducted at the classification of the calling principal. This single level classification is programmed at the VSLAN NSC. This insure peer subjects always operate at equal security classifications.

SA3: Color Changes

Calling PC users are unahle to change their current classification. If a PC user wishes to change his current classification he must terminate the current session and reinitialize the VSLAN NSD with a different Datakey.

SA5: SU Prohibitions

SU is prohibited except from the system console.

SA6: Accountability

Individual users are held accountahle for their actions through detailed audit trails.

5.1.1. Supplemental NTCB Mechanisms

The supplemental NTCB mechanisms that we have added to the VSLAN NTCB are shown in Figure 5.1. These mechanisms include: TCP, IP, a VSLAN NSD device driver, a security relevant Trusted Sessions Module (TSES), a trusted login program, and the telnet server (telnetd). Functionally, we have also added TCP, IP, a VSLAN NSD device driver, and the telnet client program to the PCs hut we do not consider our PC additions part of the supplemental NTCB mechanisms for reasons explained in Section 5.1.2.

5.1.1.1. AT&T 3B2 Software

As shown in Figure 5.1, some of our NTCB mechanisms are also System V MLS kernel additions (i.e. they are also additions to the originally evaluated B1 TCSEC TCB). These additions include streams based implementations of a VSLAN NSD device driver, TCP, IP, and TSES. TSES and the VSLAN device driver cooperate to provide each other with the necessary security lahels. For client programs, TSES informs the VSLAN device driver of the appropriate lahel for the requested session[9]. Additionally, the VSLAN device driver functionally controls the NSD, labels outgoing datagrams, and counters the host impersonation threat by providing the type of source address authentication discussed in Section 4.3.3.

As of this writing, parts of these additions are being modified so as to accept only VSLAN NTCB I&A information when processing remote logins. Wben complete, tbe consequence of these additions will be tbat a remote login will require a correct Unix password and the correct VSLAN Datakey.

The remaining supplemental NTCB mechanisms, login and telnetd, are not part of the System V MLS kernel. Nevertheless, the modified login program is clearly a crucial part of the NTCB. Login is responsible for establishing a single level user session (i.e. login) at the TSES provided classification. This insures that the PCs provide a virtual terminal service at a classification equal to the clearance of the operating principal. The telnetd server is trusted to invoke the modified login program that provides this security mechanism.

5.1.2. PC Software

As noted in Section 5.1.1 functionally we have also added TCP, IP, a VSLAN device driver, and a telnet client to each of the PCs. We don't consider these additions security relevant or part of the NTCB hecause the additions neither enforce, nor strengthen the enforcement, of our stated security assertions. Additionally, we are unahle to identify a method hy which intruders can take advantage of a flawed implementation of these PC additions so as to defeat our security assertions. Our reasoning follows.

To defeat SA1, specifically that part of SA1 requiring a valid password in /etc/passwd and /mls/passwd we assume the following concerning an attack. The attacker is cleared to tbe requisite classification, is a legitimate user with an active userID, and is operating with a VSLAN principal account authorized to communicate with the 3B2. In other terminology, the attacker is attempting to violate a discretionary aspect of the security policy. Using a TCP sequence numher attack, we assume that the attacker will attempt to estahlish a connection with the telnetd server following a valid I&A performed for a different user.

Fortunately, our implementation of telnet uses a single connection for identification, authentication, and data transfer. Consequently, it is fruitless for an attacker to estahlish an additional connection if he must provide the requisite authentication information. However, the fact that our application uses a single connection does not negate all risk. Suppose that we wish to inject datagrams on the valid connection as a means of mounting a denial of service or other type of attack? To reduce the risk associated with this threat, at the 3B2, we perform the type of source address authentication suggested in Section 4.3.3. When datagrams arrive with inconsistent host and NTCB addresses the datagrams are destroyed and WARNING notices are displayed at the 3B2 system console.

[9]The inverse is true for server programs.

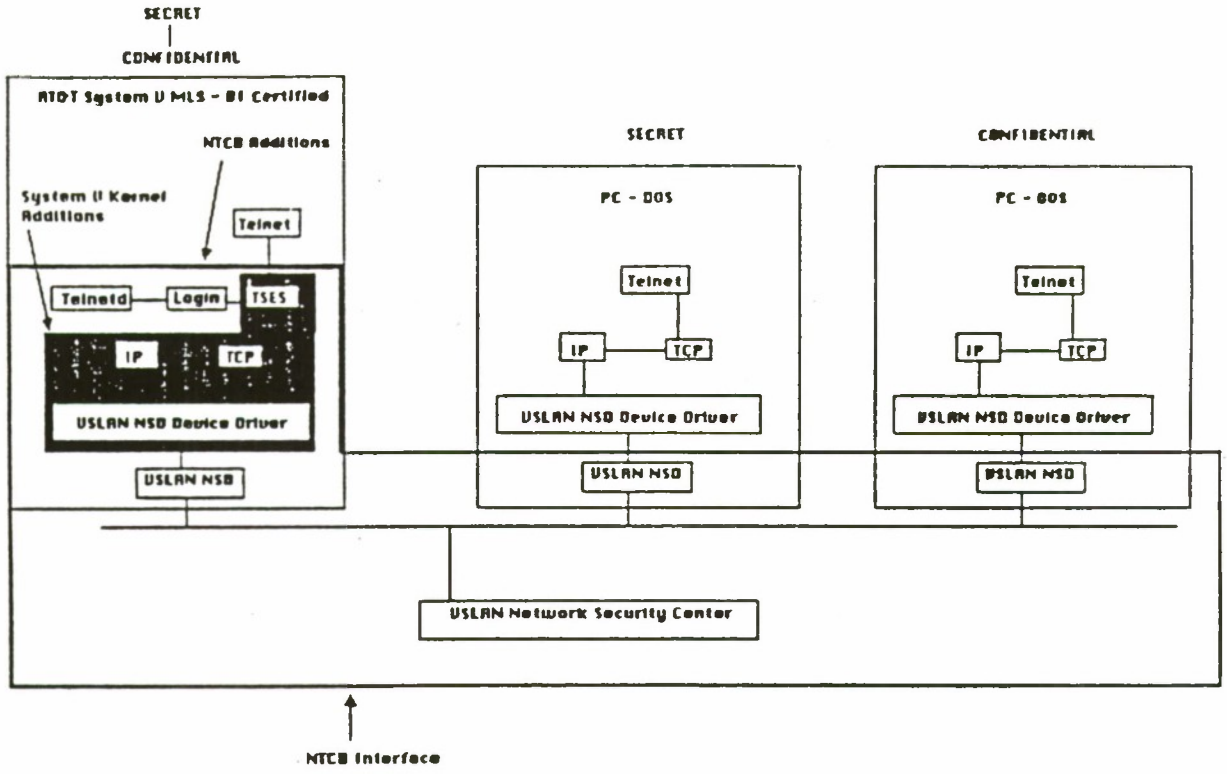


Figure 5.1. NTCB for Trusted LAN.

To defeat SA2, we assume that a legitimate user would like to establish a network connection (with the 3B2) at a security label higher than that for which be is cleared. To attempt this we assume that he would replace the existing NSD device driver with a bogus copy that labeled outgoing datagrams at the higher security label. Subsequently, because PCs are configured to operate at the single level and category set defined by the clearance of the operating principal, the datagrams output by the bogus device driver would be destroyed and audited by the VSLAN NSD. If it is unclear as to why the NSD would destroy these datagrams, the reader should review Section 1.1.

To defeat SA3, we assume that a legitimate user currently conducting a remote terminal session would attempt to use the System V MLS newpriv[10] command to upgrade his current operating classification. Such an attempt would fail because the login program initially established a single level user session at the classification specified by the VSLAN NTCB.

To defeat SA4, a legitimate user would have to establish a remote terminal session at the System V MLS label of SYSTEM[11]. Remote terminal sessions at the SYSTEM classification are prevented by procedurally requiring the NSO to correctly define principal accounts at the VSLAN NSC according to the clearance of an individual user. That is, NSOs at the VSLAN NSC are procedurally prohibited from establishing accounts at a classification of SYSTEM.

To defeat SA5, we assume that an attacker would like to destroy or otherwise modify audit trail records so as to disguise his penetration attempts. Fortunately, audit trail records are protected by the System V MLS TCB and the VSLAN NTCB. It is physically impossible to access either database over the network. For the System V audit trail, System V access control mechanisms protect the audit records. For the VSLAN NTCB audit trail, it is impossible to initiate a network connection with the NSC for the purpose of remotely modify audit records.

[10]Newpriv is a System V MLS command that allows a user to upgrade his current classification within the limits specified by the clearance range associated with the user's userID.

[11]The System V MLS TCB prevents the invocation of su except at the lowest hierarchical classification - SYSTEM. Attempts to invoke su at other classifications fail and are audited by the System V MLS TCB.

5.1.3. Operating Modes and Procedural Controls

Operating modes are implied in Figure 5.1. The 3B2 is MULTILEVEL; processing and segregating both SECRET and CONFIDENTIAL information at any given instant in time. At an given instant in time, a PC operates in a DEDICATED mode at a classification determined by the clearance of the operating principal. Depending on the operating principal and whence the associated principal's security profile defined at the VSLAN NSC, a given PC will be either SECRET OR CONFIDENTIAL. At one time, a given principal may use a specific PC to conduct a SECRET terminal session while, at a later time, a different principal may use that same PC to conduct a CONFIDENTIAL terminal session.

The trusted sessions module, TSES, and the login program are responsible for establishing user sessions at the MAC security label provided by the VSLAN NTCB component. Because network communications are always two-way, network peers always establish network connections at equal security labels to prevent security policy violations. For example, if a caller were at UNCLASSIFIED and the called were at SECRET, reads initiated by the caller would violate no read up and writes initiated by the called would violate no write down. The end result is that, despite the fact that an individual System V MLS username may be authorized for both SECRET and CONFIDENTIAL data, remote user sessions at PCs are always single level at the classification associated with the operating principal.

Finally, as indirect support for SA1 and SA2 we must impose procedural controls that limit the potential damage imposed by malicious PC programs. For example, we wish to guard against Trojan horses that might capture and store authentication information, or capture and store the results of SECRET terminal sessions. Our most potent defense against this threat is a procedural control. To restrict, and hopefully prevent, the damage imposed by such Trojan horses, we require all PC data storage to be removable or volatile and require PC users to physically secure the removable media in accordance with its classification when not in use.

6. Accreditation and the IAA Network View

We have identified many technical considerations associated with using the VSLAN NTCB as the primary foundation for a trusted local area network. In this section, we attempt to solidify some observations about issues DAAs can be expected to face.

The techniques and examples described in this paper evidence the facts that VSLAN based trusted LANs will include at least some unevaluated software and that the resulting integrations are likely to involve heterogeneous combinations that lack a uniform level of trust at all NTCB interfaces. Given that such integrations imply IAA network views, DAA attention can be expected to focus on the IAA specific issues outlined in the TNI. These issues include the interconnection rule, the cascading problem, and environmental considerations.

6.1. The Interconnection Rule

Networks accredited according to the IAA view require enforcement of an interconnection rule that limits the sensitivity levels of information that may be sent or received. This requires that multi-level devices decide locally whether information can be sent or received and requires that sensitivity labels be exchanged when information is exported from one multilevel device and imported by another. It is trivial to see that the VSLAN MAC mechanism enforces the interconnection rule. Correct enforcement depends on correct NSO inputs at the VSLAN NSC.

6.2. The Cascading Problem

The cascading problem is a situation that exists when a penetrator can take advantage of network connections to compromise information across a range of security levels that is greater than the accreditation range of any of the component systems he must defeat to do so[12]. An example of the cascading problem can be achieved by adding a B2 host and a file transfer service to our Figure 5.1 example. Assume that the added B2 host can process TS-S information and that a penetrator: (1) overcomes the protection mechanism on the B2 host to downgrade some TOP SECRET information to SECRET; (2) causes this information to be sent over the network to the 3B2 machine; and (3) overcomes the protection mechanism in the 3B2 to downgrade that same information to CONFIDENTIAL. This is the cascading problem[13]. Fortunately, after presenting the description of the cascading problem, the TNI proceeds to identify two solutions for countering the identified threat. These solutions include: the use of a more trusted system at appropriate nodes in the network[14] or the elimination of certain network connections. Assuming that mostly fiscal forces discourage the likelihood of the former solution; we concentrate our observations on the latter.

Owing to the fact that Ethernet LANs revolve around a broadcast technology, selective elimination of network connections seems hard at best when standard Ethernet based LANs are involved. Generally, a host on the Ethernet provides its network services to all other Ethernet nodes or it is disconnected from the Ethernet.

Fortunately, DAAs will note that the VSLAN DAC capability modifies the standard Ethernet broadcast technology to allow the required selective elimination of network connections. In the modified Figure 5.1 example, proper NSO configuration at the VSLAN NSC can prohibit the B2 to B1 connection that gave rise to the cascading condition while still allowing other discrete network connections to both hosts. So, in general, DAAs must carefully review proposed VSLAN DAC configurations to reduce or eliminate the threat imposed by the cascading problem.

6.3. Environmental Considerations

Concerning IAA views, the TNI states that DAAs, as a minimum, can be expected to define and document requirements for communications integrity, denial of service, and data content protection. As part of the VSLAN's evaluation as a TNI network component, the VSLAN Final Evaluation Report contains a detailed evaluation of the VSLAN NTCB with respect to these types of services. The documentation and evaluation provided there can serve as a valuable input to the accreditation process.

---

[12] TNI, Version 1, pg.249.

[13] Example nearly identical to TNI example, pg. 250.

[14] For example, replacing the 3B2 *or* the added B2 host with a B3 host.

7. Conclusions

This paper has identified technical considerations for VSLAN NTCB integrators and DAAs. The VSLAN is a B2 MDIA NTCB that integrators can he expected to supplement with additional NTCB mechanisms to form trusted local area networks. Necessary supplemental NTCB mechanisms include communications protocols, trusted operating systems, and VSLAN NSD device drivers. The current EPL population and the trend towards open computing environments suggests that the resulting integrations may require DAAs to adopt IAA views for accreditation.

References

[Bell89] Bellovin, S. M., "Security Problems in the TCP/IP Protocol Suite," Computer Communication Review, April 1989.

[Tsai89] Tsai, C., et al., "A Trusted Network Architecture for AIX Systems," Proceedings of the Winter 1989 USENIX Conference.

INTRODUCTION TO THE GEMINI TRUSTED NETWORK PROCESSOR

Michael F. Thompson, Roger R. Schell, Albert Tao and Timothy E. Levin
Gemini Computers, Inc.
Carmel, California

Abstract: This paper presents a high level introduction to the Gemini Trusted Network Processor (GTNP), briefly describing its hardware, software, network-support and multilevel security features. The general properties and intended use of the GTNP are presented.

General

The GTNP is intended to combine verified multilevel security and high performance processing to meet the Class A1 requirements of the Trusted Network Interpretation [TNI] of the Trusted Computer System Evaluation Criteria (TCSEC) for network components that implement a mandatory access control (MAC) policy as defined in Appendix A.3.1 "Mandatory Only Components (M-Components)". In addition to the GTNP TCB, the GTNP product includes functions that place the GTNP in a secure initial state: off-line administrative functions used to define those system attributes that are parameterizable and functions that validate the correct operation of the on-site hardware elements of the GTNP TCB.

Overview of Features

The GTNP TCB consists of the GEMSOS kernel and hardware base [SCHEL85], along with a non-kernel interface to support channel servers and other single-level processes. It is intended to be used as a gateway between networks of various levels, serving as an M-Component in the overall Network TCB (NTCB) architecture.

The GTNP includes a wide variety of hardware configurations ranging from the proprietary Gemini multiprocessor with eight Intel 80286 or 80386 microcomputers to the single-processor IBM PC/AT. Since it is structured to be independent of supported processors and devices, each GEMSOS hardware configuration provides logically equivalent mandatory security capabilities.

The GTNP is expressly designed with an adaptable open system architecture to support a range of network applications and to function in an embedded system. It provides a variety of disk storage and I/O device options and features dynamic configuration adaptation to the number of processors and available memory.

Applicability

The GTNP kernel has already been used in several high-assurance production systems [SHOCK88]. We have also had numerous requests from NTCB vendors for use of the product in major B3-A1 network-related projects. Our goal is to have this product certified and placed on the NCSC list of evaluated products (EPL) so that the effort required to certify it will not need to be repeated for each future project.

Architecture Overview

The GTNP is the standard commercial GEMSOS Security Kernel with single-level (untrusted) processes and a multilevel initial process (for each CPU). The initial process will be trusted over a range from system-high to system-low. It is the first process created upon booting the system, and its only function is to start single-level processes, at the levels specified by the administrator during system configuration (see Figure 1). The single-level processes are outside of the NTCB MAC partition, and as such will not need to be evaluated under the GTNP certification defined by the TNI.

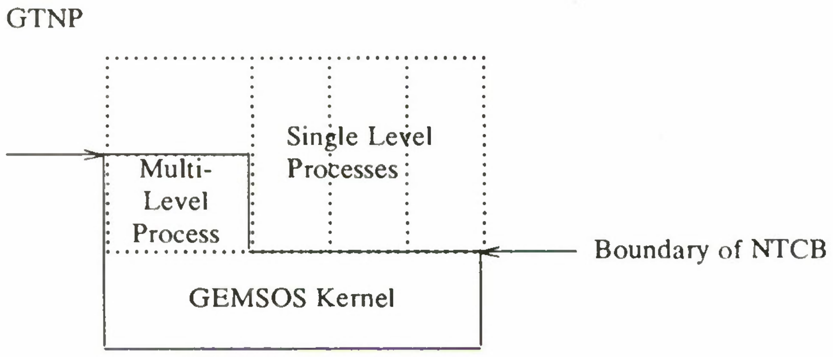


Figure 1. Network Processor Internal Architecture

It is intended that vendors building on the GTNP would replace the skeletal single-level processes that Gemini provides (for testing and evaluation purposes) with channel servers and other processes of their own (see "Coherent Network Architecture and Potential Applications," below). Depending on the functionality that vendors choose to include in them, these single-level processes may be subject to separate evaluation under the overall NTCB architecture. Changes to the single-level processes will not necessitate the re-evaluation of the MAC partition of the GTNP.

The product will provide the capability for the network manager to create other trusted processes during system configuration (e.g., to support multilevel communication channels). Use of this function will be discouraged in the Trusted Facilities Manual, because the addition of such a trusted process would likely necessitate re-evaluation of the M-component.

Coherent Network Architecture and Potential Applications

This section describes the GTNP network security architecture through the use of examples.

The GTNP could be used as a multilevel packet switch providing reliable link level communications between single-level hosts at various security levels. In addition to providing same-level communication links, the GTNP will allow reliable communication from a low-level host to a high- level host. This will entail an untrusted protocol for reliable data transfers from the low-level single-level host to the GTNP and from the GTNP to the high-level single-level host. The processes implementing this transfer protocol will be external to the NTCB MAC partition and will not be part of the evaluation of the GTNP.

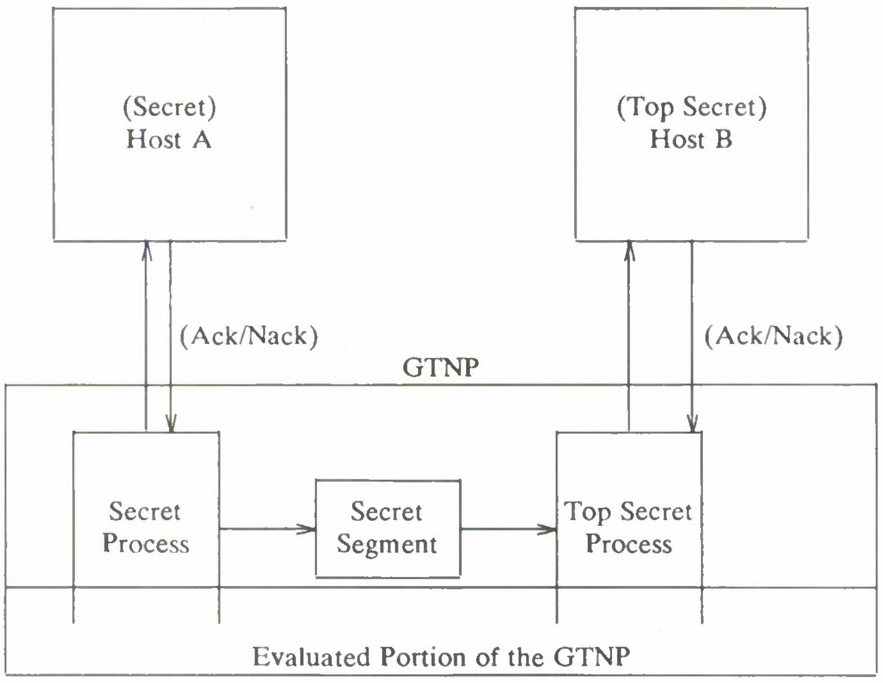


Figure 2. Example NTCB Architecture

An example showing the use of the GTNP is illustrated in Figure 2. Host A sends a message to the untrusted secret process on the GTNP. Some form of reliable protocol (acks and nacks in the example) is used between Host A and the secret process on the GTNP. This protocol is implemented in the untrusted secret process. Upon successful completion of the transfer of information from Host A to the GTNP, the message is stored in the secret segment. A signal is sent by the secret process to the top secret process on the GTNP indicating that the message is ready for transfer. At this point, the top secret process reads the message out of the secret segment and begins a transfer to Host B. A reliable protocol is used in this transfer (note that this may be the same or a different protocol than that which was used between Host A and the GTNP). The protocol is implemented in the untrusted top secret process. An acknowledgement is never sent from Host B back to Host A (because the secret process communicating with Host A cannot observe any information originating from Host B); however, the transfer is deemed reliable since the transfer of the information within the GTNP (i.e., over the bus) is deemed reliable and the transfer between the GTNP and the hosts utilizes a reliable protocol. Note that by placing the communication protocol in the untrusted processes, changes to the protocol do not require re-evaluation of the M-component.

Policy Support

The security policy of the GTNP is designed to support the the mandatory portions of the official DoD security policy (DoD Directives 5200.28 and 5200.1-R), and be applicable to many types of network configurations. This policy is incorporated into the mandatory portion of the GEMSOS formal security policy model which is based on the Bell and LaPadula model.

Both secrecy and integrity [BIBA] policies are supported, each with 16 hierarchical levels. The GEMSOS Kernel also supports 64 non-hierarchical secrecy categories and 32 non-hierarchical integrity categories. A single module, the non-discretionary security manager (NDSM) interprets the security labels. As is described below under "Replaceable Internal Modules," this NDSM can be customized to support any lattice security policy, including Clark-Wilson [SHOCK88-1] and policies needing multiple secrecy and/or integrity hierarchies or extended numbers of non-hierarchical categories.

Extensibility and Subsets

As noted in papers by Schaefer [SCHAE], and Shockley and Schell [SHOCK], if a TCB has a strict hierarchical layering it is possible to extend a mandatory-policy security kernel to support a richer set of security properties, such as those desired for the security policy of a particular NTCB. The GEMSOS kernel supports the kind of strict layering [SCHEL84] that was postulated in these papers. The Intel 80286/80386 processor used in the Gemini computer provides four hierarchical hardware-enforced privilege levels that enforce the layering. In particular, privilege level 0 (the most privileged) is devoted to the security kernel.

Additionally, the GEMSOS kernel uses the remaining three hardware privilege levels to implement a protection ring mechanism [SCHRO] that may be used to implement a program integrity policy [SHIRL] in which each process contains up to eight rings. Though the limited number of hardware privilege levels requires that a given process only have 3 active rings at a time, different processes may have different active rings and a given process may alter which of the 8 rings are currently active.

Evaluatability

The key to the Class A1 evaluation of a mandatory network component (such as the GTNP) are the formal security policy model and the Formal Top Level Specification (FTLS). The kernel of the GTNP has undergone intense scrutiny at the A1-level mandatory security. Its FTLS has been proven to support the mandatory portion of the GEMSOS formal security policy model, and was verified as providing A1-level mandatory security.

Additional Interfaces

The GTNP does not support direct user connections. Therefore, there are no interfaces for Identification and Authentication (I&A) of users or trusted path mandatory functions (e.g., changing session level). Since the GTNP does not act on the behalf of any given user (i.e., it has only internal subjects) and the GTNP will be configurable such that it has no covert storage channels (based on the analysis of the GEMSOS kernel [LEVIN]), there will not be any auditable events or audit records produced by the M-component while it is in operation. Furthermore, since there are no audit records produced during runtime, there is no runtime interface for returning audit records. Security administration (i.e., device labeling) is also handled off-line as part of system generation and does not require a runtime interface.

Hardware Configurations

With the Gemini hardware base, up to eight 80286 or 80386 microprocessors can be connected to the Multibus I to provide high throughput performance. A variety of storage and I/O devices are supported by means of interface boards connected directly to the Multibus. The system supports selected combinations of up to four Winchester and floppy disks drives. Under GEMSOS control, all processors share the connected devices.

The IBM PC/AT version of the hardware base is the standard IBM commercial product (or selected clone) modified to run the GEMSOS kernel software. This configuration includes both fixed-disk Winchester and 1.2 megabyte floppy disk drives, and the Enhanced Graphics Adaptor.

Replaceable Internal Modules

The strict loop-free layering and modular internal structuring of the GEMSOS kernel provides isolation of I/O drivers, the non-discretionary security manager (NDSM), and other internal components. This isolation, along with the concise definition of security requirements for new I/O components, permits compatible devices to be added to a GTNP configuration without affecting the integrity of the overall system.

This modularity and isolation also allows the NDSM, which is responsible for the interpretation of a given security policy, to be replaced with a similar component to support different security policies, while similarly maintaining the integrity of the overall system.

Multilevel Security

The 80286/80386 hardware supports segmented memory as well as hierarchical privilege levels for protection and mediation of all memory and I/O references. The GEMSOS kernel takes full advantage of this support.

All information stored in the GEMSOS kernel is contained in discrete logical objects (segments). Each segment possesses static attributes such as security access class and process-local attributes such as access mode (e.g., read, write, execute). Access classes are composed of a secrecy component as well as an integrity component both of which may be used to enforce non-discretionary (mandatory) security policies.

Processes are also assigned access classes. In a manner dependent on the security policy of the particular installation (see "Replaceable Internal Modules," above), process access classes are compared to segment access classes whenever access to data is requested.

Hardware privilege levels are used to further control access to information by partitioning each process into four distinct protection domains. The kernel, which mediates access to information, resides in the highest privilege level (level 0). The non-kernel TCB functions reside in the outer levels.

Encryption

Additional security support is provided by the hardware encryption device for the NBS standard DES algorithm. Each system has a unique master key and system identifier used in ensuring the trusted distribution of GEMSOS releases, in controlling unauthorized copying of system software, and encrypting the information stored on removable storage media such as floppy diskettes. Encryption can also be used by customer applications to prevent unauthorized access to transmitted data and to authenticate the integrity of received data.

Development Environment

The GEMSOS hardware base provides a self-hosting environment for software development through the use of the UNIX(tm) System V operating system. The developer uses Metaware compilers and UNIX tools for cross development following this general pathway: edit the source with a UNIX editor; compile the modules using the C or Pascal compilers; assemble any modules using the UNIX assembler; and link the modules with the UNIX linker. The result of fully resolving all references is an output file which can be exported to the GTNP environment for execution.

Concurrent Computing

Depending on the hardware configuration, the GTNP is capable of multiprocessing as well as multiprogramming. The GEMSOS security kernel can multiplex processes onto a single processor. The kernel is distributed to support combinations of parallel and pipeline processing.

Gemini's approach to concurrent computing does not require a specialized concurrent programming language, but rather uses well-developed sequential programming languages in conjunction with calls to the GEMSOS security kernel. The GEMSOS synchronization calls manipulate objects called "eventcounts" and "sequencers" to support communication and synchronization among processes [REED]. Sequential language programs use these calls to coordinate concurrently executing activity.

References

[BIBA] Biba, K.J., "Integrity Considerations for Secure Computer Systems", ESD-TR-76-372, MITRE Corporation, Bedford, MA, April 1977

[LEVIN] Levin, T., Padilla, S., "Covert Storage Channel Analysis of the GEMSOS Kernel," March 1988, Gemini Computers, Inc., Technical Report GCI-88-09-01

[REED] D. P. Reed, and R. K. Kanodia, "Synchronization with Eventcounts and Sequencers," *Communications of the ACM*, Vol. 22, No. 2, February 1979, pp. 115-124

[SCHAE] Schaefer, M., and Schell, R. R., "Toward an Understanding of Extensible Architectures for Evaluated Trusted Computer System Products," in *Proceedings 1984 IEEE Symposium on Security and Privacy*, Oakland, California, April-May, 1984, pp. 41-49

[SCHEL84] Schell, R. R., and Tao, T. F., "Microcomputer-Based Trusted Systems for Communication and Workstation Applications," in *Proceedings of 7th DoD/NBS Computer Security Initiative Conference,* NBS, Gaithersburg, MD, 24-26 September 1984, pp. 277-290

[SCHEL85] Schell, R.R., Tao, T.F., and Heckman., M, "Designing the GEMSOS Security Kernel for Security and Performance", in *Proceedings of the Eighth National Computer Security Conference,* Gaithersberg, MD, October 1985, pp. 108-119

[SHIRL] Shirley, L.J., and Schell, R.R., "Mechanism Sufficiency Validation by Assignment", in *Proceedings of the IEEE 1981 Symposium on Security and Privacy,* Oakland, California, April 1981, pp. 26-32

[SHOCK] Shockley, W.R. and Schell, R.R., "TCB Subsets for Incremental Evaluation", in *Proceedings of the 3rd Aerospace Computer Security* Conference, 1987, American Institute of Aeronautics and Astronautics, Washington, D.C.

[SHOCK88] Shockley, W.R, Tao, T.F, and Thompson., M.F, "An Overview of the GEMSOS Class A1 Technology and Application Experience", in *Proceedings of the Eleventh National Computer Security Conference,* Gaithersberg, MD, October 1988, pp. 238-244

[SCHOCK88-1] Shockley, W.R., "Implementing the Clark/Wilson Integrity Policy Using Current Technology", in *Proceedings of the 11th National Computer Security Conference,* Gaithersberg, MD, October 1988

[SCHRO] Schroeder, M.D., and Saltzer, J. H., "A Hardware Architecture for Implementing Protection Rings", in *Third Symp. on Operating* Systems Principles, October 1971, Association for Computing Machinery, pp. 42-54, 1971

[TNI] *Trusted Network Interpretation of Trusted Computer System* Evaluation Criteria, NCSC-TG-005 Version-1, 31 July 1987

AN OVERVIEW OF THE USAFE GUARD SYSTEM[1]

Lorraine J. Gagnon

Logicon Inc.
Operating Systems Division
4010 Sorrento Valley Blvd.
P.O. Box 85158
San Diego, CA 92138-5158

ABSTRACT

The U.S. Air Forces in Europe (USAFE) Guard system [1,2] provides a multilevel secure electronic interface between a Top Secret/Sensitive Compartmented Information (TS/SCI) Department of Defense Intelligence Information System (DoDIIS) Intelligence Data Handling System (IDHS) site and Secret level unit support systems connected to the Intra-theater Intelligence Communications Network (IINCOMNET). The system interfaces with the IDHS host via the USAFE Tactical Air Intelligence Network Local Area Network (UTAIN LAN), and interfaces with the IINCOMNET wing support systems via Defense Secure Network #1 (DSNET1), the Secret subnet of the Defense Data Network (DDN). The system supports the automated release of sanitized threat information, formatted in the Integrated Data Base (IDB) Transaction Format (IDBTF), and textual messages at the collateral level in accordance with Defense Intelligence Agency (DIA) policy as defined in Enclosure 8 of DIA Manual (DIAM) 50-4 [3].

INTRODUCTION

The purpose of this paper is to describe the current implementation of the United States Air Forces in Europe (USAFE) Guard system. A "guard" controls the flow of information between systems operating at different security levels. This paper summarizes the functional capabilities of the USAFE Guard, citing its unique features and describing its current status.

Why a Guard is Necessary

The "guard" concept provides a solution to a common multilevel security (MLS) problem which has existed for many years in traditional, "system high" operating environments. In order to access a system in this environment, all users must be cleared to the highest classification level of the information being processed by the system. In most cases,

---

1 This work was sponsored by the U.S. Air Force, Rome Air Development Center (RADC) in cooperation with the Space and Naval Warfare Systems Command (SPAWAR) under contract number N00039-83-C-0144.

however, only a small subset of this information is classified at the "system high" level, resulting in increased operational costs and processing overhead associated with releasing information classified lower than "system high". At this time, there are a limited number of MLS operating systems and Data Base Management Systems (DBMS) available to address the problem of multilevel information.

A guard can be used to pass information between systems operating at different security levels. Users at lower classification levels can obtain the less sensitive information available on the "system high" system through the guard. Since the guard protects against inadvertent disclosure, the "system high" system can be utilized more efficiently and cost-effectively. Thus, the guard provides a solution which is available today.

Evolution of USAFE Guard

The USAFE Guard (or Guard) project was established by the Air Force as a result of the need to send sanitized, releasable data, derived from a variety of sources, to the unit level support systems. This information consists of two distinct categories: threat data and mail. In addition, there is a requirement for mail to be sent from the operational units to the intelligence production centers.

The origin of the USAFE Guard project is based on the software architecture developed by Logicon, Inc. for the Navy under the Advanced Command and Control Architectural Testbed (ACCAT) Guard program [4]. The ACCAT Guard design was the result of over a decade of research and development in the area of multilevel secure systems. It demonstrated that the ability to connect systems at different security levels was indeed feasible. The ACCAT Guard supports the ability to send mail and perform database queries, and provides a facility for the sanitization of this information. A trusted process responsible for downgrading the information was formally modeled and verified. ACCAT Guard, which operates on the Kernelized Secure Operating System (KSOS), was installed in a testbed environment at the Naval Ocean Systems Center (NOSC) in San Diego, CA. and has been demonstrated on numerous occasions.

Initially, the USAFE Guard was intended to be developed on KSOS. However, due to an emphasis on using Commercial Off The Shelf (COTS) software, the operating system base was changed to Security Enhanced VMS (SE/VMS)[2]. As a result, the Guard can be installed on the full range of VAX processors.

CURRENT IMPLEMENTATION OF THE USAFE GUARD

The USAFE Guard system provides a multilevel secure electronic interface between a TS/SCI DoDIIS Intelligence Data Handling System

[2] DEC, VAX, MicroVAX, VAXstation, VMS and SE/VMS are trademarks of Digital Equipment Corporation.

(IDHS) site and Secret level unit support systems. The unit level systems are connected to the Guard via the Intra-theater Intelligence Communications Network (IINCOMNET). The Guard system interfaces with the IDHS host (or High host) via the USAFE Tactical Air Intelligence Network Local Area Network (UTAIN LAN). The Guard interfaces with the IINCOMNET wing support systems (also known as the Low hosts) via DSNET1 (the Secret subnet of the DDN).

The types of information that flow through the Guard, the use of secure operating system features, screening capabilities, network interfaces, auditing, and user interaction are discussed in the following paragraphs.

Transaction Types

The purpose of the Guard is to support the automated release of sanitized threat information and textual messages at the collateral level in accordance with DIA policy as defined in Enclosure 8 of DIAM 50-4 [3]. The release of labeled Secret level information residing on the TS/SCI High host is accomplished by a downgrade transaction that permits the labeled Secret information flow to the Secret network.

Three specific types of transactions are processed by the Guard:

- o High to Low Threat Update Message (TUM) Transactions
- o High to Low Mail Transactions
- o Low to High Mail Transactions

Each of these transaction types is discussed in detail below.

High to Low TUMs

Threat data is collected from many sources and stored in a Model 204 database on the IDHS host. As new threat data is received, it is formatted in the IDB Transaction Format (IDBTF), reviewed by a Security Officer and encapsulated with header information and a Cyclic Redundancy Checksum (CRC) integrity seal trailer. The header information includes the classification of the TUM, the network host name of the originator, the list of destinations for sending the TUM, a message sequence number and the transaction type (i.e., TUM). The format of the header and the CRC trailer are identical to the IDBTF format, which identifies the name of the field, a "\" character, the field value, and a "\" character (e.g., "From\Smith@HighHost\").

After the TUM has been reviewed and authorized for release by the Security Officer, it is sent to the Guard system via the UTAIN LAN using the File Transfer Protocol (FTP). After arrival and registration at the Guard, the transaction is screened to ensure that the information satisfies the releasability criteria that has been defined.

It is then released to the Low side of Guard and sent to one or more Secret level destinations on the IINCOMNET via FTP. Transactions can be addressed to a group of hosts using a "group list". The Guard will expand this "group list" and send the transaction to those hosts which are defined in the group. The Low side of the Guard can retransmit a transaction if a host is not responding.

High to Low Mail

High to Low mail transactions are handled similarly to the TUMs. They originate at the High host, where they are reviewed by the Security Officer. The header on a mail transaction contains the originator and destination addresses as part of the mail header. Following a blank line, the Guard Mail Header contains the classification, precedence of the mail (priority or routine), and the transaction type (i.e., mail). Other data may also be included, such as the name of the releaser, the date and a subject. A CRC integrity seal is placed at the end of the message. These additional header and trailer lines are formatted according to the DDN standard, with the field name, a ":" character, and the field value; white space within a line is allowed (e.g., "Classification: SECRET").

The mail is then sent to the Guard via the UTAIN LAN using the Simple Mail Transfer Protocol (SMTP). The Guard receives the mail transaction, screens it, releases the transaction and sends it to the appropriate destinations via SMTP.

Low to High Mail

Low to High mail is created at the Low hosts and sent to the Guard over the IINCOMNET using SMTP. It is not necessary to include a Guard header, although it is recommended that the "advisory classification" of the mail and its precedence be specified.

Guard accepts the transaction and sends it to the appropriate High destinations specified in the "To" field using SMTP over the UTAIN LAN. No screening is performed, except for the validation of the originating host address.

Transaction Processing

Multiple transactions are processed by the Guard concurrently. The TUM transactions are designated as the most important transactions, (i.e., they should be processed through the system at the highest priority). In order to support this requirement, the Guard provides for a set of prioritized queues (generally one per process). When a process is ready to handle a new transaction, it obtains the transaction from the queue in the following order: TUM transactions, mail transactions designated as "priority", and mail transactions designated as "routine".

Security Features of SE/VMS for USAFE Guard

In order to provide a system which meets the security requirements for accreditation, the SE/VMS operating system is used by USAFE Guard. The primary features utilized by the Guard are described below.

Separate Security Domains

The Guard file system is divided into two (2) security domains, one High and one Low. Transactions arriving at the Guard from the UTAIN LAN (High side) are placed into the High security domain until they have been screened and it has been determined that they can be released to the Low side of the Guard (and hence the Low users). Similarly, transactions arriving from the IINCOMNET to the Low side of Guard are initially placed in the Low security domain until they are upgraded to the High domain by the Guard. This separation is important in order to ensure that the data is properly handled by Guard.

Downgrade Privilege

In order for a file to be written from a High classification level to a lower one, the program responsible for the downgrade must first acquire the downgrade privilege. The privilege is removed after the downgrade occurs. This forces the downgrade of data to be centralized in a single location.

Password Management

All logins are managed by SE/VMS. This operating system supports the Password Management Guidelines [5] published by the National Computer Security Center (NCSC). It controls user logins to Guard, verifies the password and audits all login attempts.

Screening Capabilities

The Guard provides an automated screening capability for all information flowing from the High to the Low hosts. Each transaction is registered by the Guard and compared to a set of screening criteria (independent criteria exists for mail and TUMs). If a transaction satisfies the criteria, the downgrade is audited and the transaction is downgraded to the Low side of the Guard for transmission to the specified destinations.

If the information does not satisfy the established screening criteria, the transaction is rejected. The rejection and the contents of the transaction are audited by the Guard and the transaction is returned to the High host indicating that it was "rejected for downgrade". As part of the rejection handling in the Guard, there is an upper limit to the number of rejections which can occur (i.e., the "rejection limit"). If this limit is reached, the Guard does not allow any further transactions of that type into the system and does not attempt to release that type of transaction. (It should be noted that the mail and TUM transactions are handled independently so that, even if transaction processing of one type is halted, the other may still flow through the system.) A mechanism is provided to allow the rejection limit to be reset and transaction processing to be resumed.

The Guard provides several methods for validating the contents of a transaction, including:

- White space may or may not be allowed in the transaction.
- A line may be composed of a field name, delimiter and field value.
- The field value is the correct length and has the correct type of characters (i.e., alphabetic, alphanumeric, integer, decimal).
- The field value is one of a list of values (e.g., "Joe", "Jack", "John"). In this comparison, alphabetic and alphanumeric fields may be designated as case sensitive and/or with spaces significant; numeric data may have zeroes significant.
- A function defined at software generation time may be specified for validating the value of the field (e.g., verify the CRC for the transaction).
- A particular field name may be required to be part of the transaction.

If one or more of these criteria are established, but the transaction does not satisfy the criteria, the transaction is rejected.

This screening philosophy is currently being used to provide a minimal set of "sanity" checks on the data being sent from the High Host. It could easily be extended to be more comprehensive, providing a rigid set of conditions that transactions must satisfy prior to their release to the Low users.

Network Interaction

The Guard uses the DDN standard protocols Transmission Control Protocol/Internet Protocol (TCP/IP), SMTP and FTP for transferring transactions between the High system, Guard and the Low systems. FTP is used solely for transferring TUM transactions, which are viewed strictly as file transfers, while mail transactions are sent and received via SMTP.

Two independent sets of network support software, each operating at different classification levels under SE/VMS, provide an additional degree of protection in the system. For the High interface, Communication Machinery Corporation (CMC) software is used to communicate between the UTAIN LAN and the Guard. Wollongong software is used for Low communication between the Guard and IINCOMNET.

The current system configuration allows simultaneous connections to multiple hosts on both the High and Low networks. A set of Guard application processes is provided to manage these connections and handle hosts which, for outgoing connections, are not responding. The processes route transactions to an "alternate" host if the destination host is unavailable for a designated period.

This alternate addressing capability provides a great deal of flexibility to the Guard. The Guard system administrator may define a maximum number of retries to each host and the period of time to delay between these retries. If a host is temporarily not accepting new connections, the Guard attempts to send the transaction only "n" times; it then attempts to send the transaction to an alternate host (up to three (3) alternates may be designated). If a host appears to be down, the transaction remains enqueued to that host without being redirected to the alternate. A maximum time period that a transaction can remain enqueued in the system before it is returned to the Security Officer (for TUMs) or the originator (for Mail) is specified so that transactions enqueued to unavailable hosts are eventually released by the system.

Auditing Capabilities

The auditing capabilities of the USAFE Guard are pervasive throughout the system, since data is being downgraded (released) from a High to Low security level. The Guard must create and maintain an audit trail which contains a complete record of the security relevant events. The design objectives for the system's auditing capabilities include:

- Mandatory audit events.
- Optional audit events, which can be toggled on and off interactively.
- An Exception Log containing a synopsis of the most significant audit events, to be used for a quick review of the system status.
- Ability to review the audit data.
- Ability to archive and retrieve audit data from disk and tape.

Of the 45 events which were determined to be auditable, 13 events are mandatory. Examples of mandatory events include transaction downgrade, transaction rejected for downgrade, and modifications to the screening criteria. The set of mandatory events is defined when the software is generated.

Each audit event always includes a date and time stamp. It may also include the following information, depending on the specific event being audited: the transaction identifier, type, and sequence number, the Guard user generating the event, a qualifying condition on the event, if additional data should be audited with the event, and if the data should be placed in the Exception Log.

The audit data and Exception Log may be inspected at any time. Three (3) different levels of detail can be specified for the output of each audit event. In addition, the data which is viewed may be selectively chosen by specifying a time period, a user name, a specific set of audit events, the types of transactions and/or a specific transaction identifier. In this way, both full and condensed audit listings can be made available.

The audit data file is "rolled over" to a backup data file periodically: at a user-specified time each day, when the size of the file exceeds the user-specified maximum, and when the user requests that the file be rolled-over. Since the audit data remains on the disk, the ability to archive, and also restore, this data is necessary. The Guard provides the ability to archive data to another disk or to a tape. The data may also be restored to the disk from an archived audit tape if, for example, the audit events for a specific period must be reviewed.

User Interface

Another design objective of the USAFE Guard was to minimize the amount of user intervention required. Also, the user interface should be simple and straightforward. In response to these objectives, the Guard User Command Interface (GUCI) was developed. It provides a user-friendly menu-driven interface with an easy to use help facility. When there is no user logged onto the Guard, a "monitor" is active, which indicates the current activity level of the system and signals any unusual activity or problems via audible alarms from the terminal.

There are two (2) types of Guard users. The Guard Administrator (GA) handles the administrative duties, setting up the system tables and performing other system administrative duties; these activities are not envisioned to require a great deal of system interaction after the system has been accredited. The Guard Operator (GO) is responsible for controlling and monitoring the daily operations of the system, including the system startup and shutdown, status monitoring and audit data handling. Examples of the types of commands supported by the Guard include:

- o Information on each transaction active in the system and statistics on the total and current number of transactions processed, processing time and other statistical information.
- o A continuous monitoring of Guard status, including active transactions and network activity. This monitor is active if no user is logged onto the Guard terminal, or may be selected by the GA or GO.
- o Commands to allow the audit criteria to be modified, the audit data to be inspected and audit data archival and restoration.
- o Commands to allow the screening criteria to be defined and installed on the system.
- o An interface to modify information about the hosts and the tunable system parameters.
- o A mechanism to start and stop the network activity.
- o Commands to reset the system if the screening rejection limit (the maximum number of transactions which can be rejected) is

reached and to remove transactions which are queued for screening when the rejection limit is reached.

- o An interface to allow the data collected for the UTAIN LAN to be uploaded to the LAN's Network Manager Station.

- o A procedure to shut down the Guard.

Extensibility of the USAFE GUARD System

The USAFE Guard system is designed to be modular and portable, so that a variety of users' needs can be met. The Guard was developed to provide a specific solution for the needs of the U.S. Air Force. However, due to the attention to extensibility in the design and implementation, the system could be tailored for use in a variety of other applications where a Guard is needed. It is easy to extend the architecture in order to accommodate different network protocols, new types of information flowing between the High and Low systems, and additional data screening requirements. Furthermore, the system is built upon the DEC MicroVAX-II and its SE/VMS operating system, providing an excellent migration path to smaller, more powerful and cost-effective systems, if additional processing capabilities are needed in the future.

CURRENT STATUS

The USAFE Guard is currently installed on a MicroVAX-II in a testbed environment located at Rome Air Development Center's (RADC) Multilevel Security Technology Laboratory in Rome, New York. This testbed simulates the configuration that exists in the European theater. The High host is a VAX system connected to the Guard via an Ethernet (rather than a UTAIN LAN). The Low hosts use the X.25 ROMENET to simulate the IINCOMNET network interface. The Low hosts, which are VAXstation IIIs and PCs in-theater, have been configured as VAXstation IIIs at RADC; one of these has an identical configuration to an IINCOMNET Wing host.

As a result of the preliminary testing at RADC, it was demonstrated that all transaction types (i.e., High to Low TUM, High to Low mail and Low to High mail) could be sent through the Guard. A portion of the test procedures for the Guard have been successfully executed. As part of Logicon's continuing support to RADC, the test procedure validation will be completed and the Guard will be installed in the Intelligence Information Processing Laboratory (IIPL) at RADC. Certification of the Guard is expected during 1991. Following certification at RADC, the Guard will be installed at two (2) sites in the European theater.

Future plans for the USAFE Guard include migration to a Compartmented Mode Workstation (CMW) [6] platform and enhancements to the TS/SCI IDHS-side interface by incorporating the DoDIIS Network Security Information Exchange (DNSIX) functionality necessary to operate as a compartmented host on Defense Secure Network #3 (DSNET3), the TS/SCI subnet of DDN.

CONCLUSION

The USAFE Guard system provides an opportune solution to this common multilevel security problem. It will dramatically decrease the amount of time needed to transmit information between locations which are at different security levels, especially when one considers the current air gap bypass techniques presently in use. It is also a flexible system which can support a variety of information flows, making it useful in a broad range of applications. In addition, the user interaction has been minimized, which further reduces the overhead costs associated with the handling of classified information. The USAFE Guard is an ongoing software project to solve the MLS problems of today.

REFERENCES

[1] Logicon, Incorporated, "USAFE Guard Computer Program Design Specification (PDS)", San Diego, California, (Draft), November 1988.

[2] Logicon, Incorporated, "USAFE Guard Computer Program Performance Specification (PPS)", San Diego, California, (Draft), August 1988.

[3] Defense Intelligence Agency, "Security of Compartmented Operations (U)", Defense Intelligence Agency Manual 50-4 (Confidential), Washington, D.C., June 1980.

[4] J. Woodward, "ACCAT Guard System Specification (Type A)", MITRE MTR-3634, MITRE Corp., Bedford, Mass., March 1985.

[5] Department of Defense, "Password Management Guideline", CSC-STD-002-85, April 1985.

[6] P.T. Cummings, et al., "Compartmented Mode Workstation: Results Through Prototyping", in Proc. IEEE Symposium on Security and Privacy, April 1987.

MUTUAL SUSPICION FOR NETWORK SECURITY

Ruth Nelson, David Becker, Jennifer Brunell, John Heimann
GTE Government Systems
100 First Avenue
Waltham, MA 02254

Abstract

A practical approach to network security must be based on the assumption that the network cannot be totally controlled or totally secure. We have developed a conceptual model called Mutual Suspicion to address this assumption. The elements of this conceptual model are firewalls to limit damage caused by failure of a security mechanism, local enforcement of access control policies, identification and authentication as the basis of correct access control decisions, and network-based auditing to provide better information about an intruder's activities. The mutual suspicion concept supports heterogeneous security policies and mechanisms, examples of which are given in this paper. The model also allows a local evaluation of the risk of attaching a computer system to a network and of allowing that computer to communicate through the network with another computer system.

Introduction

In order to be realistic and useful, a network security framework cannot assume perfect operation of each component of a network, of each security mechanism and of each system operator. The security approach must limit the damage caused by compromises or failures, and must provide adequate audit information for detection and analysis of security failures. We have developed a conceptual model for network security which is designed for the reality of the large, uncontrollable, world-wide internetwork. Our model is based on the assumption that the amount of trust placed in the communications network and in each of the remote computers on the network should be minimized. The goal is to model a system in which any security compromise can cause only limited damage, because the elements which control system resources are mutually suspicious. The mutual suspicion concept includes identification and authentication as prerequisites and limiting factors for access control. The concept also allows the network to support multiple definitions of security services and policies for processing systems and for network communications.

Basis of the Model

Network security requires more than access control rules which must be correctly enforced by a trusted computing base. A sound conceptual model of network security must also deal with the reality of large, heterogeneous networks, in which multiple policies may exist. The model must also address the finite probability of compromise by outsiders, by users, and by operations personnel who have physical access to the network processing and communications components. Real network components and algorithms fail, and network configurations and traffic change unpredictably. Real people do not always behave according to their security clearances. In real networks, passwords get guessed or stolen, cryptographic keys get lost or stolen, and trusted system operators may sell secrets. The network security framework must, of course, include enforcement of access control rules. It must also include limitation of damage caused by failure or compromise of security mechanisms and strong auditing mechanisms to detect and locate penetrators where logically possible. Our mutual suspicion model addresses these concerns. It is based on the following premises:

- **Components and operators are not perfect.**
- **"Trust" is not an absolute -- it is a measure of risk.**
- **Failures will occur and damage must be limited.**
- **Networks and computers are heterogeneous in function and policy.**
- **Communications connectivity is constantly changing in an uncontrolled manner.**
- **Local users are more easily monitored and controlled (hence more trustworthy) than remote users.**
- **Administrative control of a network is important for maintaining security.**

The model requires that each resource owner control access locally. Access control decisions are made according to the resource owner's local policy, and access privileges are based on the authenticated identification of the requester of the resource. Network resources include processing and communications functions as well as data. The extent to which identification and authentication are required is a function of the resource owner's policy. The means of providing the authenticated identification depends on the configuration of the network path between the requester and the owner as well as on the authentication mechanisms.

A network security concept which simply allocates security functionality to trusted components or trusted computing systems is vulnerable to compromise of a component or system. While a degree of trust is necessary to allow data communications and resource sharing in a network, the trust must be limited to the minimum required. It is risky to design a network which assumes absolute trust in any single mechanism, component or person; such assumptions can lead to global compromise. The principle of least privilege must apply to systems as well as individual users of those systems.

In the concept of computer security embodied in the Trusted Computer System Evaluation Criteria (TCSEC)[1], users of the system, or rather the software processes which operate on their behalf, are not trusted. The users and their software are assumed to require constraints on their activity (through the reference monitor) so that they will not violate security policy. The Trusted Computing Base (TCB) controls users' access to the computer resources, and includes all the trusted software in the system plus the hardware base on which it runs. However, the TCB is actually only part of the trusted computing environment. Trust in the TCB depends on physical control of the computer room environment and personnel security control of the system operators. Trusted computer systems do have audit requirements, but these may be undermined by an operator with or without later detection. Since access to the computer room is physically controlled, and since the operators are employees of the system owners, accountability is feasible and the trust is reasonable.

There is a problem in extending this trust to a network situation, as is done by the Trusted Network Interpretation (TNI)[2] of the TCSEC. While the distributed TCB (the network TCB or NTCB) may be correct and enforce the system security policy, the NTCB itself is not sufficient. System security depends on the physical and operational control at all of the computer facilities in the network. If any of these sites is compromised, then the network may be compromised. The distributed trust model described in the TNI is vulnerable if any of the pieces is vulnerable, and physical compromise cannot be completely addressed by software security mechanisms. Allocation of security functionality to trusted network components does not change the assumption of physical and operational security at all the participating sites. The partitioning may in fact make security more difficult, since it allows more heterogeneity and since the allocation may be done improperly or weakly. For example, if auditing and access control are done at separate network components, then a failure of either component or of the communications medium may prevent the auditing of security-relevant events.

From a practical standpoint, the TNI approach is limited to small, essentially stable networks, with a single administration, so that each computer system containing part of the NTCB is provided with the required physical and operational security. The mutual suspicion concept addresses the network security problem in a way that allows network sites to have heterogeneous security environments.

Mutual Suspicion and Access Control

In a system based on mutual suspicion, each resource owner (where the resource is computing or communications capability, or data) acts on the principle of least privilege to protect its own resources. Redundant security checks, carried out by separate resource owners, form "firewalls" which prevent single failures from compromising large portions of the network. Access decisions are made on the basis of authenticated identity of the resource requester, and access is restricted if there is authentication uncertainty. User access control and auditing are done in the context of the network, utilizing path information as well as source information. Figure 1 illustrates a network with numerous firewalls protecting its communications and computing resources. It shows a user/server model for simplicity; in most cases the identification and authentication function is bidirectional.

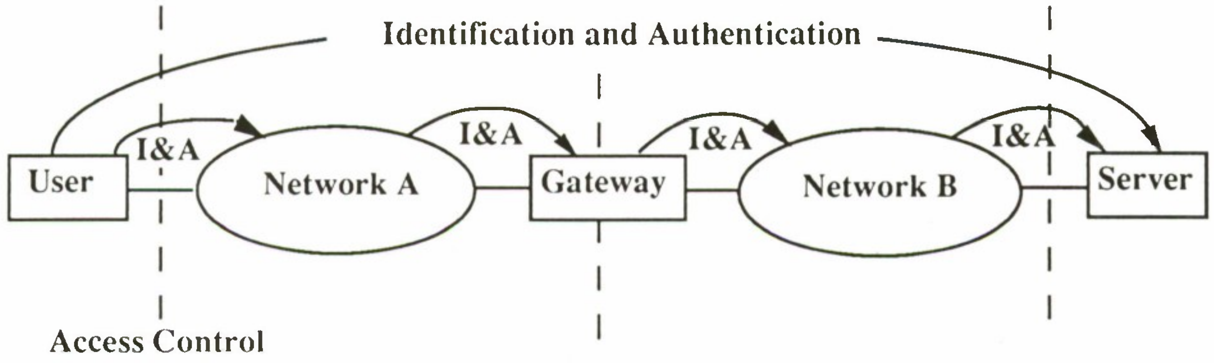


Figure 1. In a mutually suspicious internet, each resource owner makes independent access control decisions based on the authenticated identity of the requester.

As an example, suppose the user on Network A wished to access the server on Network B. First, he (his workstation or terminal) would have to access network A. This requires that the network know who is requesting access (for example, because the interface is hard-wired) and that the requester be allowed to send packets across the network. Next, the user's communication needs to traverse the gateway, which may be physically attached to network A (so that the gateway knows what network the communication is coming from), but which may also make access control decisions based on security label, source and destination end-system addresses, or even user ID. Network B then makes its access control decision to allow the packet to enter. The server makes its decision on whether to allow the communications and the path is established. However, the user must still satisfy the server's access control policy in order to access the server's processing or data resources. The server may require a user login, password, etc., so that it can base its decisions for access to these resources on more specific information than that used for the communications. At each step, the resource owners implement their own access control decisions, based on their policies.

This example shows the operation of firewalls. At each step, the resource owner can block further access by the user. The access takes place only if all resource owners on the communications path and the destination server concur. This provides additional safety from the viewpoint of access control. The access control policies must, however, be consistent and practical to provide access to legitimate network users. In addition, if service assurance is needed, multiple paths through the network must be provided so that failure of a single component does not cause denial of service.

Each resource owner can independently determine the information it needs to make such a decision. The information may be carried implicitly (e.g., in the physical connectivity) or explicitly (e.g., in a security label on a packet). The amount of delay and computation required for the access control decisions will depend on the individual component policy and on the physical and logical design of the network.

Access Control Policies

Each resource owner is responsible for enforcing its own policies. These policies are administratively determined. Effective control of network usage requires functional limitation of access. This is in addition to the mandatory and discretionary access control described in the TCSEC. The functional limitations are needed to enforce the security principle of least privilege and to prevent misuse of resources and compromise of data.

The security policy of a computing system in the internet can vary depending on its purpose. For example, a system whose function is to distribute advertising information accepts queries from any user of the internet. However, it accepts new advertisements only from validated advertisers who have active accounts and who can be trusted to pay. This system accepts all incoming communications requests and all advertising queries, but it requires extensive identification and authentication before allowing changes to its advertising database.

A very different example is provided by a multilevel secure system. This system restricts communications to security levels within its accredited range, based on packet security labels which are trusted not to change during communications. Additionally, the system requires identification and authentication of individual users so that it can enforce its mandatory and discretionary access control policies.

Network service providers can also implement a variety of policies. For example, a network provider may provide access control by limiting physical attachment. Any user with a physical connection may use the network. This is often true for host attachments to packet-switched networks through dedicated lines.

In other networks, a validated security label may be required for access, and this label may be constrained to a set of permissible values. This is the type of policy enforced in a multilevel secure network.

In still other networks, particularly those with access through the public telephone network, authentication as a network subscriber may be required for access. This type of access control, with a user-provided password, is used for terminal access to the Defense Data Network through a Terminal Access Controller.

Authentication and Trust

While the mutual suspicion concept places emphasis on minimizing trust requirements, trust cannot be completely eliminated from a system which permits communications and sharing. If a computer system allows sensitive data to be sent over a communications network to another computer system, then it is trusting both the network and the remote system to some extent. The design of the systems and their operation must be adequately secure to warrant this trust. The allocation of security functionality to particular mechanisms in the computer and communications systems can affect the degree and kind of trust required of each.

Suppose that two computer systems dedicated to the same processing mission are connected to each other by a point-to-point encrypted link. The users of the two systems are all authorized access to all information in both systems. In this example, the encrypted link prevents any leakage of data outside of the two connected systems. It also provides authentication of each system to the other (with suitable key management). The systems are run in dedicated mode and so no further access control mechanisms are necessary.

Now suppose that the systems are connected through a packet-switched network rather than by a direct link. More trust is required of both the computer systems and the network. If the network encrypts traffic on all of its links, then the data is protected from disclosure outside of the network, but the switches must be trusted to prevent disclosure to unauthorized network users. If suitable end-to-end encryption is used, then the trust requirements on the network are reduced significantly, but the requirements on the computer systems increase because they must be trusted not to leak data through the unencrypted headers into the network. The end-to-end encryption provides authentication of each computer system to the other, without relying on correct delivery by the switches.

If the communicating computer systems are not dedicated to a single purpose, but instead are supporting users with different access privileges, then the systems must trust each other to enforce these access limits. This means that the communicating computers must know not only each other's identity but also each other's access control capabilities. It is not sufficient for one computer to authenticate a user on a remote computer. If the remote computer does not provide sufficient access control protection, then it may give one user's data to another, unauthorized user.

End-to-end encryption can provide very strong authentication of two computers to each other. The encryption key is a form of firewall, in that it limits the damage that can be caused by failures in the network or in computer systems which do not hold the encryption key. However, it is also important to control the flow of data through the computers themselves. If data is sent to an untrustworthy computer, then the data may be propagated to any other computer which can communicate with the untrustworthy computer. For this reason, it is important to observe the principle of least privilege and to limit communication between computer systems to that which is necessary and authorized. Strong identity-based access controls can define communities of computer systems which have reason and authorization to communicate. Additional mandatory, discretionary and functional access restrictions can reduce the risk of this communication.

Authentication Uncertainty and Access Control

Access control to each resource is required for security, but correct access control decisions must be based on authenticated identification. Our model requires strong authentication of the requester before full privileges are granted, and allows only limited privileges if the authentication is too weak. For example, a computer system might limit a requester's access privileges to the intersection of the privileges of the requester and those of all other users of the network through which he accessed the computer system. This would be an appropriate choice in the case where the network does not ensure user authentication, since the resource owner cannot be sure which network user really made the request. If the network authenticates the requester's source system (host or terminal access device), the privileges could be somewhat more generous -- for example, those which are common to all users of the source system. If the network authenticated the source system and the source system was trusted to authenticate its users and maintain the security of their data, then the requester's rights alone could determine his access privilege. In summary, authentication uncertainty limits access permissions:

- **If the user is authenticated, then he gets his access privileges.**
- **If only the user's host is authenticated, then the user gets only the privileges available to all users on that host.**
- **If only the network connection is authenticated, then the user gets only the privileges available to all network users.**

In a secure network or system, "superuser" access and diagnostic access must be strictly limited, either to local users or to very strongly authenticated users from authorized and specified sites. These types of accesses can and have subverted software security mechanisms. A recent computer break-in exploited a weakness which allowed an "anonymous" user to change his identity to superuser, while using an unchecked password for the anonymous account.

In the mutual suspicion concept, the identification and authentication function acts as an outer ring of protection around the computing or communications resource, as shown in Figure 2. Access control decisions cannot be completed until the identity of the requester is sufficiently authenticated, with the required granularity of identity and the degree of authentication dependent on individual system policy. Once the authentication is complete, the system has the information needed to enforce its individual access control policy. Uncertainty of authentication logically requires limitation of the user's access, if the policy is to be effectively enforced. Authentication and identification protection can be added to a secure system to refine, not violate, its original access control policy model, whether that model is Bell and La Padula[3], Clark and Wilson[4], or any other. The COMPUSEC-based access control models define access rights of known users; the identification and authentication function provides assurance that the user is indeed known.

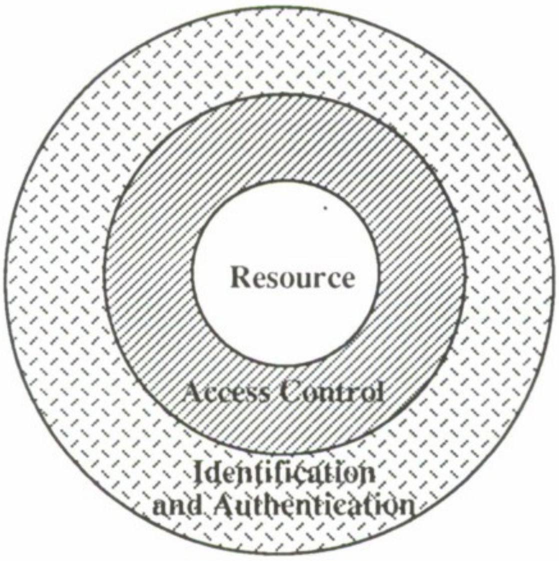


Figure 2. The identification and authentication function is the outer ring of protection around the computing or communication resource.

A network which enforces the mutual suspicion model provides a double ring of protection against penetration attempts. The penetrator has to break through the protection mechanisms which separate authorized users of the systems and associated security levels, but he must first defeat the identification and authentication protection to get access to the system at all. If the connecting systems distrust each other (as they should if their evaluation classes are low or their authentication mechanisms are weak) then the penetrator's access rights to the connected system will be downgraded to a safe minimum. If this minimum is outside the intersection of the ranges of both systems, no connection will occur.

Controlling the Risk of Network Attachment

In a large network, the number of potential interconnections and users increases the risk that data will be compromised. In particular, users with *no* authorized access to a system may attempt to break into a target system, and they may use their own computing resources to do so. The sensitivity of data and the amount of data in a system increases its value as a target and increases the risk to the system and the harm an attacker could cause. If the network is constructed without firewalls, then the network as a whole becomes a large and attractive target. There have already been break-ins in which the attacker penetrated a weak system (e.g., by guessing passwords) and then used the resources of that system to penetrate other computers on the network.

It is difficult to control the connectivity and configuration of a large network, since any of the attached computers can change its configuration or add "back-end" attachments in a way which is invisible to the network administration. While there may be administrative rules against such unauthorized modifications, they are enforced locally by system operators who may or may not all be trustworthy and competent. Therefore, if the security of the network depends critically on the correct operation of each of a known set of network components, then there is no way of evaluating or controlling risk.

The use of the mutual suspicion model for a secure network allows risk to be evaluated more locally, since each resource owner bears much of the responsibility for its own protection. For example, consider a host computer system attached to an internet through an end-to-end encryption device which provides mandatory access control. The risks in this system would be primarily related to the probability of failure of the host to enforce discretionary access control, the probability of failure of the host to enforce mandatory access control within its accredited range, and the probability of misidentification of one of its users over the network. The TCSEC evaluation class of the host as a computing system gives a measure of its strength in these areas. The requirement for authentication and the limitation of access privileges by authentication uncertainty also limit system risk and allow its evaluation.

If a user is sufficiently authenticated by the network and has the access privileges to be acceptable to the connected system, then the risk is reduced to that described in the TCSEC and environmental guidelines: he is a known user with known access rights, and those are within the accreditation range of the system. Without mutual suspicion and user authentication protection, *any* access may present a risk outside the accreditation bounds of the system, since it may be made by a user with lower clearance than the minimum required for authorized system use.

Control and Auditing of Network Paths

Our network model assumes that failures will occur despite all of the security mechanisms designed into the network. Firewalls are useful to limit damage from these failures; auditing is necessary to detect failures and identify the sources of attacks. A system which collects information about how users access computers through a network can help in the tracking effort. In addition, if this information is available to the access control function at the time a user tries to access a system, then it can be used to help determine the authentication uncertainty and limit suspicious accesses.

If the user accesses the computer system through a network, the system can derive the identities of the link, the network and the remote host or terminal access system used by the user, and the user's ID, from network protocols. This information is available at the time of login and can be used to help make access control decisions based on access path plus identity. Use of this information can be implemented locally, without a change to the network protocols, but it does mean that the access control and user identification functions of the computer system must be closely linked to its network protocol functions.

There have been a number of break-ins in which the unauthorized user logs into a weakly protected computer, steals a user identity and password, and then leapfrogs via remote login to a remote computer. In order to control and trace this type of activity, an access control function should be added to the virtual terminal protocol so that path information about the host systems is forwarded along with user ID for remote logins. Figure 3 shows the forwarding of this information. This path information would allow a destination system to make access control decisions based on more host-to-host segments of a user's path, ideally back to his local terminal. If the path is too long, passes through suspect systems, or is not sufficiently authenticated, access could be limited or denied. This access control protocol requires the computer system at the origin of the remote login request to have stored the path segment(s) from the user's terminal to that point. Forwarding of the ID and path information can be done via a virtual terminal protocol at connection establishment.

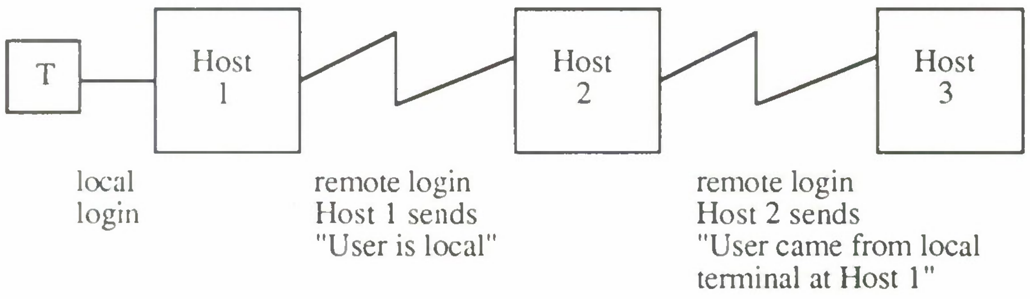

T
Host 1
Host 2
Host 3
local login
remote login Host 1 sends "User is local"
remote login Host 2 sends "User came from local terminal at Host 1"

Figure 3. Forwarding of access path information can help in access control and auditing.

Network-based auditing is a crucial part of a secure network. For example, if a computer permits remote logins, then it should audit information about the user's access path, derived from the network protocols and/or from a secure virtual terminal protocol. The path information provided by the protocol can be used to make access control decisions (as described above) and can also provide an audit trail to detect and localize network security violations. In recent break-ins, path auditing would have allowed the intruders to be traced easily and quickly, rather than with the great effort that was actually expended. Even partial information, such as the identity of the distant host, gateway or terminal access controller, would have sped the process.

Conclusions

The network environment presents unique security problems which cannot be solved on a global basis. Robust security in a real, large, heterogeneous, dynamically changing network requires that each computing system must be responsible for protecting itself and its resources. Each computing system must limit reliance on external information to that received from reliable sources with authenticated identities and established rights. Since the network configuration changes dynamically, security must not depend on the global properties of the network but rather on the characteristics of the communicating computing systems and the specific path between them. The effects of damage and compromise should be limited. The mutual suspicion model of network security embodies these requirements.

The mutual suspicion model requires risk to be controlled at each system and so it allows risk to be evaluated locally. It provides a basis for evaluating network security which is consistent with the distributed way in which networks are developed and administered.

We are now working to develop the model further by identifying security functionality required of the computing systems and network service providers and determining methods of achieving and evaluating high assurance of this functionality.

Acknowledgements

The work described in this paper was funded by the National Computer Security Center under Contract Number MDA904-89-C-6030. The authors of this paper gratefully acknowledge the comments and insights of many experts, both inside and outside the Center, who helped us during the development of the concept as well as with review of the paper. The views and conclusions expressed in this paper are those of the authors and should not be interpreted as necessarily representing official policies of any organization.

References

1. Department of Defense, "Department of Defense Trusted Computer System Evaluation Criteria," DoD 5200.28-STD, December 1985.

2. National Computer Security Center, "Trusted Network Interpretation," NCSC-TG-005, 31 July, 1987.

3. D. Bell and L. La Padula, "Secure Computer System: Unified Exposition and Multics Interpretation," MTR 2997, The MITRE Corporation, Bedford MA, July 1975.

4. D. Clark and D. Wilson, "A Comparison of Commercial and Military Computer Security Policies," in Proceedings of the 1987 IEEE Symposium on Security and Privacy, Oakland California, April 1987.

A SECURITY POLICY FOR TRUSTED CLIENT-SERVER DISTRIBUTED NETWORKS

Russell Housley
Sammy Migues
Xerox Special Information Systems
7900 Westpark Drive, Suite A210
McLean, VA 22102

ABSTRACT

Many of today's network products are based on the client-server distributed network model. Our goal of implementing a class B1 trusted network led us to discover that while many of the concepts we required were in existence, they were very scattered. This situation required us to develop a security policy defined at the subject and object level, a more security-conscious definition of client-server, and a discussion of the NTCB partitions and their sufficiency to provide a network reference monitor. This paper describes the results of those efforts.

1.0 Introduction

Large computer networks are needed to meet today's information processing needs; however, these networks are rarely designed with sufficient security features. The client-server distributed network model[1] is being used in more and more of these large networks (most often, they are actually internets). The work in this paper represents a step towards implementing a class B1[2,3] client-server network.

We started by defining a security policy which draws from previous work by many other people. For example, the mandatory access controls are derived from the work done by Bell and LaPadula[4] and the mandatory integrity controls are derived from the work done by Biba[5]. The resulting security policy includes mandatory, discretionary, and transmission policies for secrecy and integrity. It also includes supporting policies for audit, identification and authentication, and object reuse.

The second step was to formalize the characteristics of client-server distributed networks. We published the security policy and the client-server characterization in the proceedings of the Fifth Annual Computer Security Applications Conference[6]. We received much feedback from this paper and the comments were used to update and improve the work. [Thank you to all who took the time to review and comment on our previous paper.]

Finally, we present a small argument showing that our NTCB partitions are capable of representing a unified network reference monitor. We wish to stress that while the model of an NTCB partition is somewhat specific to our project, the client-server definition and security policy were designed to be useful to others working in this area.

This work has only recently been completed. We present it here hoping for additional feedback from the audience.

2.0 Security Policy

To facilitate the design of a trusted distributed client-server network, a network-oriented security policy was developed. This security policy includes mandatory secrecy, mandatory integrity, discretionary secrecy, and discretionary integrity policies for protecting data in components as well as transmission secrecy and transmission integrity policies for protecting data in transit. Supporting policies for identification and authentication, audit, and object reuse are also included.

The following definitions apply:

1. Secrecy label A is dominated by secrecy label B if the hierarchical secrecy level in A is less than or equal to the hierarchical secrecy level in B and the set of non-hierarchical categories in A is contained in the set of non-hierarchical categories in B.

2. Integrity label A is dominated by integrity label B if the hierarchical integrity level in A is greater than the hierarchical integrity level in B and the set of non-hierarchical categories in B is contained in the set of non-hierarchical integrity categories in A.

We realize that Appendix B of the Trusted Network Interpretation (TNI) refers to the goal of an organization's security policy as controlling the access of people to data and that the policy can be stated without the use of jargon. However, such a high-level statement was of limited use to our project and, consequently, the level of abstraction used below was chosen.

2.1 Discretionary Access Control Policy

2.1.1 Discretionary Secrecy

No subject shall be able to read or execute any object protected by the NTCB unless granted explicit permission by a subject with such authority over that object. A subject shall be able to read or execute objects only through the proper use of the appropriate NTCB interface protocols.

2.1.2 Discretionary Integrity

No subject shall be able to modify any object protected by the NTCB unless granted explicit permission by a subject with such authority over that object. A subject shall be able to modify objects only through the proper use of the appropriate NTCB interface protocols.

2.2 Object Reuse

No storage object shall contain any data for which a subject is not authorized when that storage object is allocated or reallocated to that subject.

2.3 Marking Policy

The marking policy assertions are as follows:

a. All subjects and all objects readable by subjects external to the NTCB shall be labeled. Clients (subjects) shall be labeled at creation time with the label requested by the user if the label is allowable for the subject, for the workstation, and for the communications channel. The label for a newly-created object shall dominate the label of the creating subject.

b. Labels shall not change during the life of the subject or the object.

c. Label integrity shall be maintained while labeled objects are in transit.

2.4 Mandatory Access Control Policy

2.4.1 Mandatory Secrecy

No subject shall be able to read or execute any object protected by the NTCB unless the current secrecy label of the subject dominates the secrecy label of the object. A subject shall be able to read an object protected by the NTCB only through the proper use of the appropriate NTCB interface protocols.

No subject shall be able to modify any object protected by the NTCB unless the secrecy label of the object exactly matches the current secrecy label of the subject. A subject shall be able to modify an object protected by the NTCB only through the proper use of the appropriate NTCB interface protocols.

No subject shall be able to create any object within the NTCB unless the secrecy label of the created object dominates the current secrecy label of the subject. A subject shall be able to create an object in a container protected by the NTCB only through the proper use of the appropriate NTCB interface protocols.

2.4.2 Mandatory Integrity

No subject shall be able to read or execute any object protected by the NTCB unless the current integrity label of the subject dominates the integrity label of the object. A subject shall be able to read or execute an object only through the proper use of the appropriate NTCB interface protocols.

No subject shall be able to modify any object protected by the NTCB unless the current integrity label of the object dominates the integrity label of the subject. A subject shall be able to modify objects only through the proper use of the appropriate NTCB interface protocols.

No subject shall be able to create any object within the NTCB unless the current integrity label of the subject dominates the integrity label of the created object. A subject shall be able to create an object in a container protected by the NTCB only through the proper use of the appropriate NTCB interface protocols.

2.5 Identification and Authentication Policy

No subject shall be able to access any resource controlled by the NTCB without successfully authenticating its identity to the NTCB partition providing that resource.

2.6 Audit Policy

The NTCB shall be capable of auditing all security-related events.

2.7 Transmission Policy

2.7.1 Transmission Secrecy

User data in a protocol data unit shall be protected from disclosure to any subject (authorized or unauthorized), except for the originator and the intended recipient(s), while the protocol data unit is in transit from the originator to the intended recipient(s).

2.7.2 Transmission Integrity

User data in a protocol data unit shall be protected from undetectable alteration by any subject (authorized or unauthorized), except for the originator and the intended recipient(s), while the protocol data unit is in transit from the originator to the intended recipient(s).

2.8 Trusted Subjects Policy

Trusted subjects shall be able to violate only the mandatory and discretionary access control policies and only through methods which are both controlled and auditable by the NTCB.

3.0 The Client-Server Model

The formalized client-server distributed network model is described by the following properties:

a. (1) Clients shall be the entities which request resources from services through application layer communications protocols. Clients may or may not be NTCB partitions.

(2) Services shall be Network Trusted Computing Base (NTCB) partitions which perform high-level functional activities on behalf of a client (See Figure 1). A server shall be the physical means (hardware) by which a service performs its functional activity. A service shall act as a client when it makes a request for resources from another service. This shall only occur when it requires resources other than, or in addition to, the ones it provides to fulfill a request made by the original client.

(3) A service may be allowed to violate the mandatory and discretionary policies, but only within its own partition and it shall still be considered suspicious by all other partitions.

(4) A conversation shall be a mutually-authenticated association between a client and a service. Conversations shall maintain the security of the information transmitted between clients and services.

(5) Services shall be stand-alone or distributed. Stand-alone services shall be those in which multiple instantiations of a single service do not cooperate. Distributed services shall be those in which multiple instantiations of

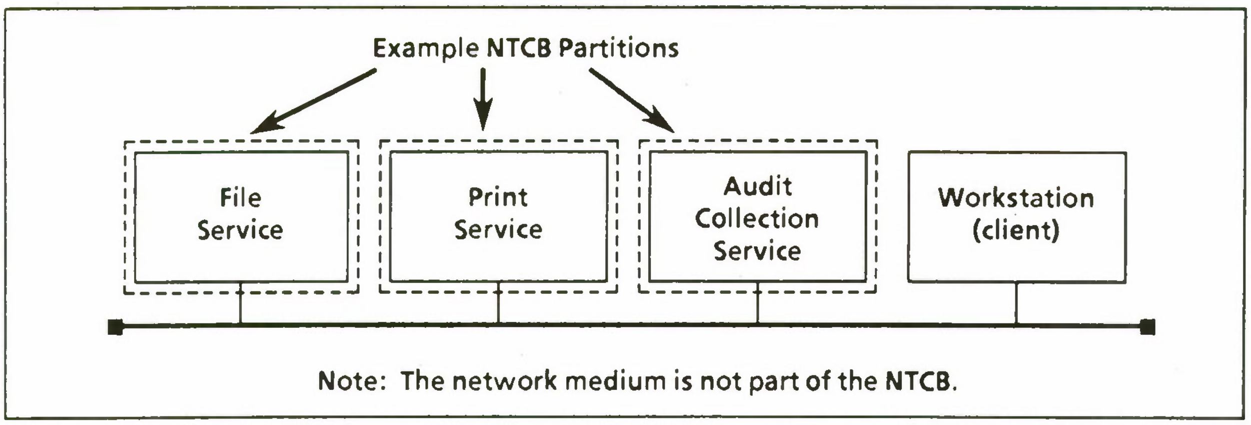


Figure 1. Example NTCB Partitions.

a single service cooperate actively to provide a unified service.

b. Clients and services shall be the only subjects and all subjects shall be uniquely named. The address space of a subject is confined to a single NTCB partition throughout the subject's lifetime.

c. All NTCB interfaces shall be described by well-defined application layer communications protocols.

d. NTCB partitions may allow other subjects to read portions of their address space.

e. Services shall protect the resources (e.g., a file system, printers, dial-up channels, etc.) they provide to the network by only allowing requests to enter through the NTCB interface (the application layer protocols).

f. All instantiations of a distributed service must process the same label range. Stand-alone services may process any range of labels from one (i.e., single-label) to all (i.e., network low to network high).

g. The NTCB partition shall contain a reference monitor which shall ensure that clients pass all requisite mandatory and discretionary access control checks before access to resources is allowed.

4.0 NTCB Partitions

The individual services of the distributed network correspond very well with the concept of partitions in an NTCB. Each partition (service) is responsible for protecting only those resources which it provides to the network. Figure 1 shows examples of NTCB partitions.

For example, a file service provides file storage and retrieval resources to the network clients. It will also provide the protection for the file system in the form of identification and authentication, discretionary access control, mandatory access control, object reuse, and audit.

The identification and authentication is performed when the client initiates a conversation with the service. The client and the file service are mutually authenticated.

The identification information from conversation establishment is used by the file service to make discretionary access control decisions.

The label associated with the conversation is the basis for mandatory access control decisions. Files (or directories of files) transferred to the file service for storage will receive labels which dominate the conversation label.

As required, a file service will generate audit records for the actions it performs. The audit records will be available for later perusal by a system security officer.

5.0 Reference Monitor Argument

In this section, a short argument is developed demonstrating that the NTCB partitions in the Trusted Xerox Network Systems (XNS) project form a unified network reference monitor. All partitions are "MIAD" components as defined by the TNI. That is, components which provide mandatory access control, identification and authentication, audit, and discretionary access controls.

Note that, in Trusted XNS, the concepts of "partition" and "component" are identical.

5.1 Axioms

The standard axioms discussed in Appendix B of the TNI are as follows (some wording was changed):

1. A subject is confined to a single NTCB partition throughout its lifetime.
2. A subject may access directly only those objects within its NTCB partition.
3. Every NTCB partition contains a reference monitor that mediates all attempted accesses made by clients.
4. All communications channels linking NTCB partitions do not compromise the security of the information transmitted over them.

For Trusted XNS, we also include the following:

5. Subjects may only access NTCB partitions through the establishment of a mutually-authenticated conversation.
6. The NTCB partition interface shall be composed of a finite number of well-defined application layer protocols.
7. NTCB partitions only respond to well-formed calls to valid protocol entrypoints.

5.2 Argument

The following simple argument uses a state transition approach to show that all network accesses are mediated and that the network reference monitor cannot be tampered.

In Trusted XNS, clients shall access services through the establishment of conversations. Conversation establishment shall occur in two parts: first the client must contact the Authentication Service (an NTCB partition) and prove its identity, then the client contacts the desired service passing along credentials given to it by the Authentication Service. A proper response from the service to the client completes the mutual authentication. An attempted access made through means other than a properly established conversation is ignored.

The state where an NTCB partition has no conversations is inherently secure. The partition is trusted to manage its space correctly and there is no path for commands from any untrusted source to be entered.

The Authentication Service will only forward a credentials package to the client if the client correctly proves its identity and if the mandatory access control label requested for the conversation is appropriate for the communications channel, the client, and the service. The service shall only establish the conversation if the credentials package is valid. Therefore, the state transition from no conversations to one conversation is secure.

All conversations being held by an NTCB partition are cryptographically separated. Therefore, the state transition from one conversation to more than one conversation is secure.

The client subject does not "move" to the NTCB partition and no remote process is created. The client can only forward commands to the NTCB partition over the conversation. Therefore, there is no state transition for creating a local process.

An NTCB partition shall not attempt to access an object in another partition using the credentials of the client. Therefore, there is no state transition for non-local access of an object.

All NTCB partitions shall contain a reference monitor and the NTCB interface (the protocols) shall only forward well-formed calls to valid entrypoints to the reference monitor. All other calls shall be discarded (and auditable). Therefore the state transition from waiting for input to forwarding input to the reference monitor is secure. Also, the state transition from processing input in the reference monitor to delivering output to the client is secure.

All conversations shall be cryptographically protected therefore communications channels shall not be able to compromise the security of the information transmitted over them.

Clients shall notify NTCB partitions when a conversation is to be ended and the NTCB partition destroys its part of the conversation. If the client does not inform the partition, the partition shall destroy the conversation when the lifetime of the cryptographic key has expired. No other client can transmit over an existing conversation since the key would not be known. Therefore, the state transition of deleting a conversation is secure.

6.0 Conclusions

Overlaying NTCB partitions over the services of a distributed network is very effective. It defines a flexible architecture which is easily expanded to include new services. The notion of having each service protect its own resources reduces network overhead to a minimum and allows each service to manipulate its resources in the most efficient manner. Also, since the services form the NTCB interface, this boundary can be easily shown and the network protocols used to request resources from the NTCB become a precise NTCB interface specification.

Xerox is building upon this work and expects to develop class B1 network services which other vendors may use in their network product. The Vendor Assistance Phase (i.e., developmental evaluation) is currently underway. The first release of Trusted XNS shall not address mandatory integrity.

7.0 References

[1] "Xerox Network Systems Architecture General Information Manual," XNSG 068504, Xerox Corporation, April 1985.

[2] "Trusted Computer System Evaluation Criteria", DoD 5200.28-STD, National Computer Security Center, 26 December, 1985.

[3] "Trusted Network Interpretation," NCSC-TG-005, National Computer Security Center, 31 July 1987.

[4] D. Bell and L. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," MTR-2997, The Mitre Corporation, March, 1976.

[5] K. J. Biba, "Integrity Considerations for Secure Computer Systems," MTR-3153, The Mitre Corporation, April, 1977.

[6] Sammy Migues and Russell Housley, "Designing a Trusted Client-Server Distributed Network," in Fifth Annual Computer Security Applications Conference Proceedings, pp. 91-94.

NETWORK SECURITY AND THE GRAPHICAL REPRESENTATION MODEL*

Jared S. Dreicer, N-4, MS E541
Laura Stolz and W Anthony Smith, Graduate Research Assistants
DOE Center for Computer Security
Los Alamos National Laboratory
P. O. Box 1663
Los Alamos, NM 87545

ABSTRACT

This paper describes the underlying conceptual design and investigative approach used during the development of the Prototype Graphical Representation Model. The initial problem was to characterize and develop the fundamental theoretical foundation for modeling the features of computer networks. This research was influenced by the desire to investigate graph theoretical problems, in general, that are common to many different systems and disciplines. A computer network is a specific graph theoretical problem. This paper provides details on the early research into the relation between computer networks and graph theory and the optimal representation of computer networks for security analysis.

I. INTRODUCTION

The Prototype Graphical Representation (PROGREP) model effort is funded by the Office of Safeguards and Security at the Department of Energy (DOE) primarily to investigate security in computer networks. The PROGREP Model also includes the capability to investigate information flow in communication systems and to provide a graphical display of these communication systems and networks. At this time stand-alone computer systems are exceptional; the trend in new and modified computer systems is toward networking because it provides benefits such as economies of scale, enhanced productivity, efficient communication, resource sharing, and increased reliability [1]. Inherent in the desire to network is the implicit acceptance of increased interconnection with other computers that may also be interconnected to other unknown computers or networks. This increased connectivity can result in a combinatorially explosive number of communicating computers. Networking, however, also presents a challenge and potential disadvantages with respect to maintaining and ensuring the integrity and security of the networked computer systems. Further, networking creates a large number of other related problems, such as path routing, scheduling, network control, cycle generation, traversability, and connectivity [2-6]. Security and other problems are of particular concern depending on the classification and character of the data that are processed, stored, or transmitted on computer networks and communication systems. These issues are of particular concern to the DOE because of the sensitivity and national security nature of the data that are processed and stored on DOE and DOE contractor computer systems.

The DOE has a large number of local area networks (LANs) and subnets (small LANs connected to larger networks) and is connected to a variety of national and international networks (e.g., BITNET, HEPNET, ARPANET). DOE also operates several wide-area networks for its own use (e.g., NWCNET). For DOE contractors to perform their work efficiently, computer networks are necessary. However, the more they are needed, the more important it is to determine methodologies and procedures to ensure the network security. The following recent events demonstrate the need for applied research and development in network security: the German Chaos Club's infiltration of computer systems at various U.S. government organizations and various penetration attempts and attacks on other government organizations that are on the

*This work was supported by the U.S. Department of Energy, Office of Safeguards and Security.

INTERNET. The rapid emergence of networks has been beneficial, but network security research has just been initiated. The knowledge, tools, and capability to sufficiently understand and address the problem are in short supply. The applied research for the PROGREP model is the first step in developing a research program, tools, and methodologies to investigate network security.

Although the PROGREP effort was funded to conduct applied research into computer network security, the model appears to be applicable to many other disciplines. There are parallels between the basic graph theory principles of computer networks and systems that can be portrayed by graph structures. For example, the PROGREP model also applies to the safeguards discipline. In computer security the intent is to protect the data and information on computer systems; in safeguards the intent is to protect the special nuclear material and the inventory data related to the material. With modifications, the PROGREP model could represent special nuclear material process lines, which are fundamentally graph structures. The PROGREP model can currently represent process lines (directed graphs) but will need to be modified to characterize the real world and model specific safeguards systems.

II. PURPOSE

The PROGREP system is being developed to (1) better understand computer networks for future research and development; (2) provide a tool capable of graphically representing any computer network, which is required by computer security personnel; (3) create methodologies that detect and indicate security relevant information and events and check the security of proposed network topologies; and (4) expand the means to conduct further network security and graph theory research.

III. GOALS

The primary goals of the PROGREP research are to help system security personnel check the security of existing networks, to determine the security of proposed networks, and to conduct applied research into graph theoretical problems. Therefore, it is our goal to produce a realistic and valid network representation system, not the ultimate system. While developing PROGREP, we tried to provide a useful tool for computer security personnel. Our ultimate goal is to provide a means by which security personnel may enhance their understanding and the security of an actual computer network.

IV. PROGREP MODEL SYSTEM SPECIFICS

The PROGREP software system has been implemented on a Texas Instrument Explorer using the expert system shell called Knowledge Engineering Environment (KEE), Common Lisp methods, icons, object-oriented programming methodologies, and KEE Pictures for graphical display [7]. The PROGREP model provides a user interface that is designed to allow a user the ability to rapidly and efficiently represent graph components, their interconnections, and interrelationships.

Objected-oriented programming methodologies naturally complement the software development, result in a generalized tool, and enhance the functionality of a graph structure system. This is a result of the dependence on set theory for defining graphs and on the abstract notion of passing information (e.g., material) among vertices along edges. Objects are entities that can be described as having behavioral or cognitive capabilities (procedures) as well as physical assets and attributes (data) [8]. There are two main concepts that distinguish object-oriented programming: message passing and specialization [9,10]. Message passing is the functional essence of object-oriented programming; all activity is dependent on the "action-response" from sending messages between objects. Message passing is equivalent to a sophisticated procedure call. Specialization is the combination of data structure, class inheritance, and data hiding (due to inheritance constraints). Specialization enhances object hierarchies, data abstraction (through inheritance), and instantiation.

Object hierarchies or classes allow objects to be either exactly alike or almost alike with respect to the physical (data) and behavioral (functional) characterization of the system being modelled. Data abstraction eases the burden of data modification and input and also reduces the specification of redundant information due to the inheritance features. Instantiation uses the inheritance hierarchy to specify an individual object. The PROGREP model employs these methodologies by defining two main classes: components (vertices) and links (edges). The physical and behavioral information that is related to a particular component or link is controlled by the own/member class inheritance constraints available in KEE [7]. The PROGREP model extends the concept of object-oriented programming by the use of objects as icons. An icon is a behaviorally functional and physically characterized graphically operational object.

V. CONCEPTUAL DESIGN

The first phase of the development of the PROGREP model was to establish an analytical basis by which to generically define computer networks. An additional constraint was that the model must be flexible in representing and characterizing real-world systems (e.g., computer networks and nuclear material process lines). We imposed this requirement so that other research efforts in the Safeguards Systems Group and at the DOE Center for Computer Security at Los Alamos National Laboratory would benefit from this latitude. During this phase of the effort, it became apparent that there was no clear technical description of a computer network.

What is a computer network? Can a stand-alone computer constitute a computer network? Regardless of the answer (one could contend that a massively parallel computer is a network), is it necessary to include stand-alone representation in the PROGREP model? Are computer networks different than distributed systems? These were some of the questions we addressed during the early phases of this research. We addressed these questions in terms of the capabilities desired for the PROGREP model. Even though a stand-alone computer is not typically considered a computer network, we included the capability of representing stand-alone computers in the PROGREP model.

In PROGREP our definition of a computer network is very general. It is any collection of interconnected, autonomous computers or components of slave hardware (e.g., printers, disk storage components, or plotters). If two or more computers or components are able to exchange information, then they are interconnected. This definition of a computer network complements the definition of a graph. A graph $G = (V, E)$ is a structure that consists of a finite set of vertices V and a finite set of edges E (an edge is specified by an unordered pair of distinct vertices). In the PROGREP model, computer networks are fundamentally represented and characterized in terms of graph theory and graph structures. A network $N = (C, L)$ is a structure that consists of a finite set of components C and a finite set of links L (a link is specified by an unordered pair of distinct components). The components (computer or slave hardware) of a computer network (e.g., computer, gateway, printer, or disk storage) are defined in terms of vertices and the interconnections or network links are defined in terms of edges. These links may be either uni- or bi-directional and physical (an actual connection) or abstract (hardware data transfer compatibility but no actual connection).

In the PROGREP model, stand-alone computer security and network security requirements and limitations are modelled as constraints at the components and across the links of the represented network (graph structure) [11,12]. Typically, computer security programs depend on organization-specific policy statements. These policy statements are generally implemented by imposing constraints, procedures, and restrictions in the following areas: hardware/software security, telecommunications security, administrative security, personnel security, and physical security [13-16]. The PROGREP model addresses some of the issues associated with the above mentioned areas but is primarily a security assurance, design, and analysis system. The types of security checks addressed are related to compatibility, consistency, and suitability of hardware designations and interconnections. Additionally, the transfer of data from a source to a destination is scrutinized for the creation of a cascade problem [17], the existence of unacceptable operation modes, and other transmission path problems. Because we decided to include stand-alone computers in addition to computers connected into a network, it was natural to divide the computer network security problem into two sub-problems. One represents and characterizes the stand-alone computer security risks, and the other represents and characterizes the network security risks.

A. Stand-Alone Computer Security

Our model of the security of a stand-alone computer depends on data classification level, user clearance level, the machine's evaluated product lists (EPL) level, the operating mode of the computer, and a protection index [13-16]. The security risks on a stand-alone computer are related to computer access, data integrity, and data sensitivity. The data stored and processed on a computer are assigned a classification level which reflects the importance of protecting their integrity, that is, preventing inadvertent or intentional modification, destruction, or disclosure of the data. Users of the computer are assigned clearance levels and need-to-know permission which allows read/write access to data in the computer that have been assigned an equivalent or lower classification level. The EPL level of a computer indicates its ability to prevent and indicate unauthorized user access to data. The operating mode of the computer is either dedicated, system high, compartmented, or multilevel. The protection index depends on the user clearance level and the data classification level relative to the EPL level of the computer on which the data are stored and processed. The protection index reflects the inherent vulnerability of the data to access (i.e., highly classified data accessed by an uncleared user) on a particular computer. Using the protection index, PROGREP specifies the minimum EPL level acceptable that is needed to keep the data from being vulnerable. Because the protection index is a function of the user clearance and data classification levels, the security requirements for a stand-alone computer translate into the protection index indicating the required minimum EPL level that the computer must meet.

To determine whether or not a stand-alone computer meets its security requirements, the PROGREP model determines the appropriate operating mode and EPL level from the user responses. The algorithm that carries out the operating mode check is as follows:

(1) Determine whether all users on the machine are cleared for the highest data classification resident on the machine. If some users are not cleared for the highest data, then the machine operating mode should be Multi-level.

(2) If all users are cleared for the highest data on the machine, then determine if compartmented information exists on the machine. If no compartmented information exists on the machine, then determine if all users have a common need-to-know for all data on the machine. If all users have a common need-to-know for all data, then the machine operating mode should be Dedicated. If some users do not have a common need-to-know for all the data, then the machine operating mode should be System High.

(3) If all users are cleared for the highest data on the machine and if compartmented information exists on the machine, then determine if all users have access to all compartments on the machine. If some users do not have access to all compartments, then the machine operating mode should be Compartmented. If all users have access to all compartments and have a common need-to-know for all data, then the machine operating mode should be Dedicated. If all users have access to all compartments and some users do not have a common need-to-know for all data, then the machine operating mode should be System High.

The algorithm that implements the EPL level check is as follows [13-15]:

(1) Calculate the protection index based on the user specified data classification level, need-to-know access, and user clearance level. Note: [In Refs. 14 and 15, this protection index is referred to as the risk index, and there is also a slight indexing difference.]

(2) Determine the minimum EPL level required to satisfy the protection index.
(3) Calculate the designated machine's actual EPL level based on the types of security features (i.e., authorization, audit, and access controls) that are present.
(4) Compare the machine's actual EPL level with the minimum EPL level required (based on the protection index), and ensure that the actual EPL level is greater than or equal to the minimum EPL level.

These algorithms are also used when determining the security of a network.

B. Network Security

We have based the model of network security on an extension of the notions presented above for a stand-alone computer, i.e., data classification level, user clearance level, computer EPL level, operating mode of the computers, and a protection index. A network is composed of individual computers interconnected by links. Hence, each computer has the individual security risks concerning computer access, etc., previously discussed and the propagation of local risk [17], which is related to the possibility of a vulnerability on an individual computer propagating to one or more computers linked in the network. The propagation of local risk can cause a network vulnerability to appear as if it were a stand-alone machine vulnerability.

Therefore, one would think that a simple solution would be to collapse and treat all the components in a network as a single computer system. This would require determining the highest data classification level, the lowest user clearance level, and the resulting protection index for each component. Employing these protection indices, one would then have to determine the minimum EPL level required for every component on the network to ensure that it is secure given the worst case security requirement (low user clearance and high data classification). Having determined the applicable worst-case minimum EPL level, it would be required for all components on the network, regardless of circumstances. This is neither a realistic nor a feasible solution. It would severely diminish the benefits of operating on a network. Instead we have approached the problem from a systems perspective.

With respect to security, a network can be thought of as the combination of various subsystems. Each component and each link of a network are subsystems that have specific requirements and risks associated with them. This systems perspective permits the security features of the heterogeneous subsystems to be evaluated in terms of a homogeneous network.

The algorithms that we employed for stand-alone computers are transferable with modifications and extensions to deal with the interconnectivity inherent in networks. The major security issues that are unique to a network are the propagation of local risk and the cascade problem [13, 17]. The cascade problem is concerned with lowering the classification level of the data (downgrading) on one computer and then transferring the data to another computer at the lower classification level. These two problems make securing networks more complex because of the need to treat individual protection indices, risks, and security features from an aggregated perspective. We approached this system's problems by initially ensuring the security of the individual computers (as described in the previous section). Then when a connection (link) is created, it is assigned a maximum data classification level. This classification level is used to determine the data transfer capability of the link with respect to the specifics of the components being interconnected. Further security checks are executed to ensure that the heterogeneous components act in a homogeneous manner with respect to the network. Some of these checks address the operating mode and protocol compatibility between interconnected computers, the possible creation of a multilevel system, and the indication of a cascade problem. Briefly, the algorithm that implements the link security checks is as follows:

(1) Determine the maximum data classification level of the link.
(2) Execute a connection check to determine what is being interconnected. There are three possible cases: two links are being connected, a link and a component are being connected, or two components are being connected.
(3) Depending on the interconnection case, further checks are executed. For the link-link connection, a data classification compatibility check is executed. For the link-component connection, a comparison between the link data classification and the data classification of the component is executed. For the component-component connection, compatibility checks for operating mode, user clearance and data classification are executed, and then a cascade problem check is invoked. The cascade check implements the nesting condition test [17]. If the nesting condition test fails, a modified version of the stand-alone EPL level algorithm is executed.

The combination of all these checks ensures the security of the network or at least provides indications and warnings to a user of any security problems with the configured network. Further research has been conducted on ensuring the security of transmissions across links. Methodologies and algorithms have also been developed that allow the determination of security and constraint problems on network paths. A brief discussion of the current PROGREP model will indicate the nature of the capabilities and security features that have been employed.

VI. FUNCTIONAL DESCRIPTION OF THE PROGREP MODEL

We sought to develop a generic model that allowed security personnel to consider "what-if" questions in the computer network and security domain. New configurations, policies, protocols, hardware, software, and operating concepts are continuously developed and deployed. The ability to use these developments or encourage their use in a cost-effective manner, in part, depends on our capability to determine their operational impact on security. To determine this impact, it is necessary to configure and characterize the computer systems forming a deployed network. This allows security personnel to specify the particular security-related characteristics of their network and to then determine their network security problems or concerns. The PROGREP model provides a mechanism that intelligently directs the user to provide the necessary input and allows the user to create a display of the network configuration. This intelligent interface aids in the dynamic network creation by providing logical control of the specification of the computer characteristics and security factors through the use of text and graphics. There are two major steps in the network representation process: building and displaying the network and related information. Both functions are carried out by menus activated by mouse buttons.

A. Network Display Functions

Five display menus correspond to and are named for the five objects that appear in a network: a network, a sub-net, a machine, a backbone, and a link. (The same as in the construction section.) These menus are employed as described in the construction menu section. The hierarchy of menus and menu functions is as follows:

Display Menus

Network Menu	Sub-Net Menu	Machine Menu	Backbone Menu	Link Menu
Attributes	Attributes	Attributes	Attributes	Attributes
Magnification		Transmit Msg	Transmit Msg	
Scroll				

B. Network Construction Functions

Five construction menus correspond to the five types of objects that can appear in a network: a network, a sub-net, a machine, a backbone, and a link. Each menu references more menus, which are called up in the following ways. The Network Menus are called up by clicking the mouse (left or right) while pointing the mouse at the background. The Sub-Net Menus are called up by mousing on a Sub-Net Circle. The Machine, Backbone, and Link Menus are called up by mousing on a corresponding object on the screen. The hierarchy of menus and menu functions follows:

Construction Menus

Network Menu	Sub-Net Menu	Machine Menu	Backbone Menu	Link Menu
Add Node	Delete	Add Link	Add Link	Label Link
Load Network	Move	Add Node	Add Node	
Save Network	Pop Sub-Net	Clone Machine	Delete	
View Up	Push Sub-Net	Delete	Move	
	Rename	Move	Push Sub-Net	
	View Down	Push Sub-Net	Remove Link	
	Remove Link	Rename	Rename	
			Resize	

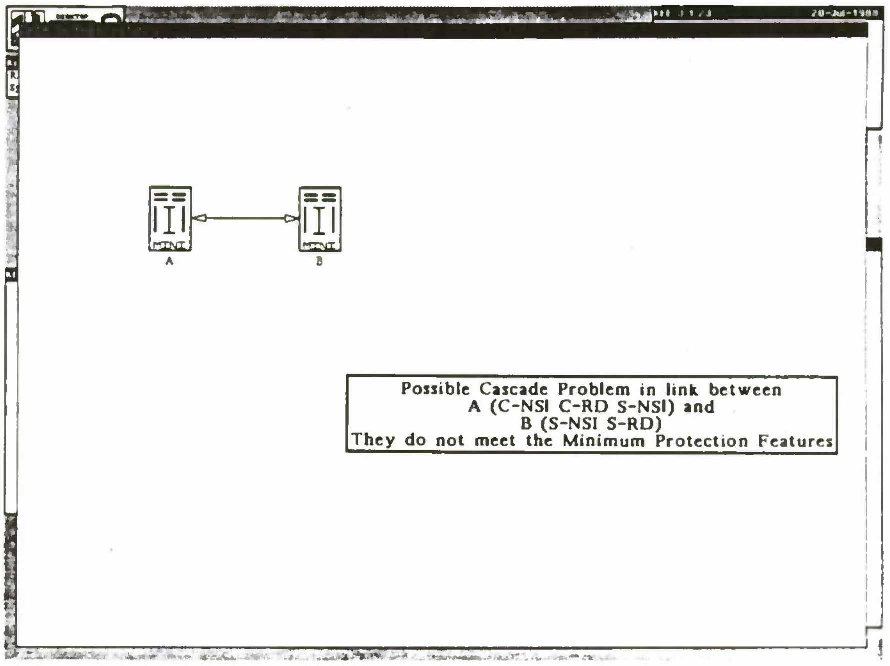
A simple example of the type of graphical representation for a computer network that the PROGREP model is capable of analyzing and displaying is presented in the next section. The displayed network is tailored after the Integrated Computer Network (ICN) at Los Alamos National Laboratory but is by no means an exact duplication.

C. Example Network

An example network will be presented that demonstrates the graphical nature and some of the security checks and other features that are executed in PROGREP. The example will be given in three related steps; the first step is associated with interconnecting two stand-alone computers, the second step is an extension of the first by connecting a computer to one of the two existing computers through a backbone connection, and the third is a further extension of the network topology achieved by adding a new link between two of the three computers.

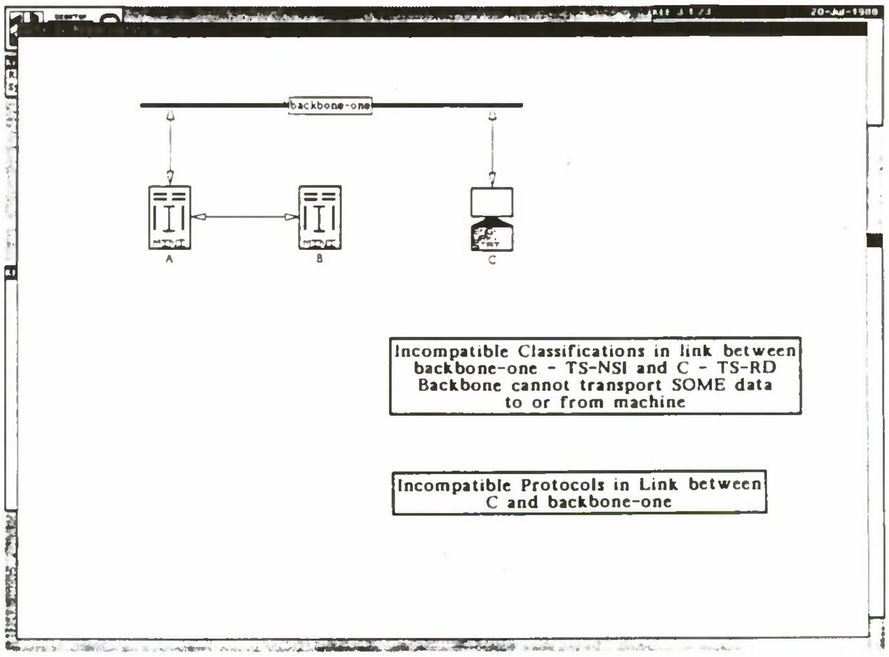
In the first step, both stand-alone computers A and B have been designated as possessing the following security features and capabilities: identification and authentication, audit trails, access controls, and both A and B have been designated as having a Multilevel operating mode and running the TCP/IP network communication protocols. The minimum and maximum data classification pairs on A and B are (C-NSI, S-NSI) and (S-NSI, S-RD), respectively. Finally, the minimum and maximum user clearance level pairs on both A and B are (L, QN). The creation of a network link between A and B generates the security warning indication of a possible cascade problem as seen in Fig. 1 because of the discrepancy in data classification levels on the computers.

In the second step, a network backbone running TCP/IP communication protocols and capable of handling a maximum data classification of TS-NSI has been created. When computer C is connected to the backbone, several warnings are generated (Fig. 2). These result from the user designations that have been associated with C. Computer C has been designated as possessing the following security features and capabilities: identification, authentication and audit trails, but not possessing access controls, internal labeling, and assurance testing features. Further, C has been designated as having a Dedicated operating mode with all users having a common need-to-know



Possible Cascade Problem in link between
A (C-NSI C-RD S-NSI) and
B (S-NSI S-RD)
They do not meet the Minimum Protection Features

Fig. 1



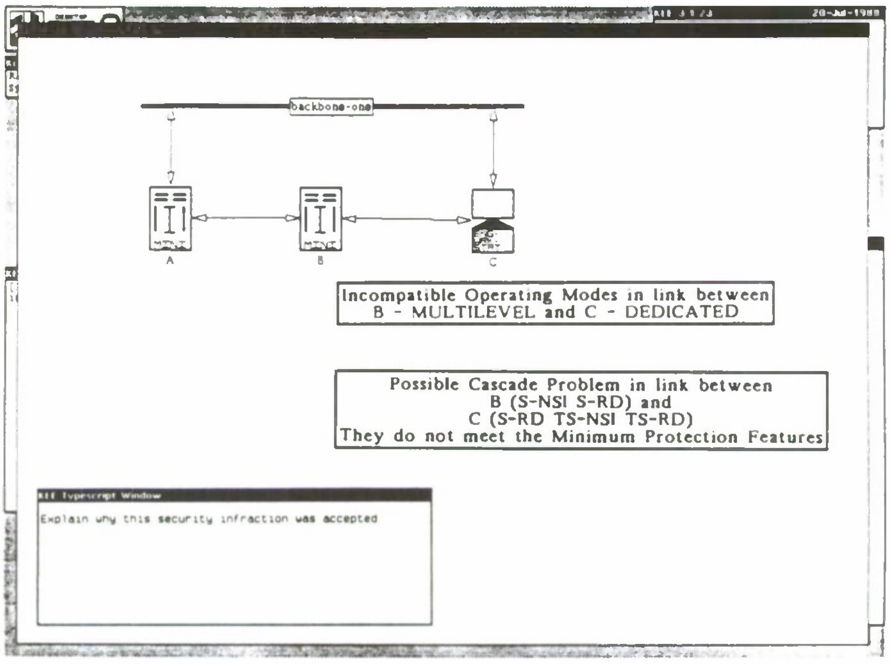
Incompatible Classifications in link between
backbone-one - TS-NSI and C - TS-RD
Backbone cannot transport SOME data
to or from machine

Incompatible Protocols in Link between
C and backbone-one

Fig. 2

and running the CHAOS communications protocols. The minimum and maximum data classification pair on C is (S-RD, TS-RD). Finally, the minimum and maximum user clearance level pair on C is (QS, QS).

Finally, in the third step, the creation of a network link between computers B and C generates the security infractions that are a result of the particular user designations. Figure 3 lists these infractions and displays the user explanation input capability.



Incompatible Operating Modes in link between
B - MULTILEVEL and C - DEDICATED

Possible Cascade Problem in link between
B (S-NSI S-RD) and
C (S-RD TS-NSI TS-RD)
They do not meet the Minimum Protection Features

```
Explain why this security infraction was accepted
```

Fig. 3

This example presents a brief and partial list of the types of response that an analyst would receive from PROGREP when configuring an actual or proposed network.

VI. SUMMARY

The PROGREP model research has provided great insight into approaching the modeling of graph structures in general and computer networks in particular. It enables the display of the components and the links of a graph structure. The PROGREP model was designed to quickly and efficiently represent network components, interconnections, and interrelationships. The main features of the PROGREP model are the flexibility of intelligent and graphical interfaces. The intelligent interface aids the user in the dynamic network creation by providing logical control of the specification of the computer characteristics, parameters, properties, and security factors through the use of text and graphics. The graphical interface allows the user to display the topology of the configured network and analyze its security.

Several approaches are taken to answer network security issues. The first approach is the stand-alone security checks and data capture. These security checks ensure compliance with policy concerning the use of various operating modes and the necessary hardware and software functions associated with particular EPL levels. The second approach is the systems perspective relative to network interconnection security checks and data capture. These security checks ensure the data transfer compatibility over a link, the operating mode compatibility between components, the indication of the creation of a multilevel system, and the indication of a possible cascade problem between components. It also supports the investigation of information flow problems and constraints through the message transmission capabilities of PROGREP. The combination of all these security checks is essentially equivalent to those required in DOE Order 5637.1 [13] and those described in Part I and Appendices A, B, and section of C of the Trusted Network Interpretation [17].

A third approach is currently being developed. It incorporates the integration of network security services into the existing PROGREP model. These additional features will model the functionality of the ICN at Los Alamos and will be essentially equivalent to Part II of all of Appendix C [17]. Other future work will be to develop and incorporate simulation capabilities, to enhance and expand the existing explanation features of the system, and to continute the network intrusion detection research that has been initiated. Currently, collaborative efforts between Los Alamos and the University of New Mexico has resulted in the prototype network level monitor [18]. We believe that these enhancements will provide the ability to address most network security and information flow problems.

REFERENCES

[1] A. S. Tanenbaum, Computer Networks. New Jersey: Prentice Hall, 1988.

[2] M. F. Capobianco, M. Guan, D. F. Hsu, and F. Tien, Eds., "Graph Theory and Its Applications: East and West," in Proceedings of the First China–USA International Graph Theory Conference, New York Academy of Sciences, 1989.

[3] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco: Freeman, 1979.

[4] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Great Britain: Wiley and Sons, 1985.

[5] T. Nishizeki and N. Chiba, Planar Graphs: Theory and Algorithms. Amsterdam: North-Holland, 1988.

[6] R. J. Wilson and L. W. Beineke, Applications of Graph Theory. London: Academic Press, 1979.

[7] J. S. Dreicer and D. Topkis, private communication July 1989-October 1989, discussions related to network security and the cascade problem, in collaboration with Los Alamos National Laboratory.

[8] D. Topkis, private communication October 1989, draft paper "The Cascade Problem for Multi-Level Security in Computer Networks."

[9] "KEE Reference Manual," Intellicorp, May 1987.

[10] R. Fikes and T. Kehler, "The Role of Framed-Based Representation in Reasoning," Communications of the ACM, Vol. 28, No. 9, pp. 904-922, 1985

[11] J. F. Sowa, Conceptual Structures – Information Processing in Mind and Machine. Massachusetts: Addison-Wesley, 1984.

[12] M. Stefik and D. G. Bobrow, "Object-Oriented Programming: Themes and Variations," AI Magazine, Vol. 6, No. 4, pp. 40-62, 1986.

[13] "Classified Computer Security Program," DOE 5637.1, Department of Energy, January 1, 1988.

[14] "Computer Security Requirements–Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments," CSC-STD-003-85, Department of Defense, Computer Security Center, June 25, 1985.

[15] "Technical Rationale Behind CSC-STD-003-85: Computer Security Requirements–Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments," CSC-STD-004-85, Department of Defense, Computer Security Center, June 25, 1985.

[16] "Department of Defense Trusted Computer System Evaluation Criteria," CSC-STD-001-83, Department of Defense, Computer Security Center, August 15, 1983.

[17] "Trusted Network Interpretation," NCSC-TG-005, Department of Defense, Computer Security Center, July 31, 1987.

[18] J. S. Dreicer, A. B. Maccabe, G. Luger and D. Topkis, private communication since April 1989, discussions related to network level monitoring and application of intrusion detection techniques [genetic algorithm and neural networks].

TESTING A SECURE OPERATING SYSTEM

Michael Johnston and Vasiliki Sotiriou

TRW Systems Integration Group
One Space Park
Redondo Beach, CA 90278

Abstract

Assuring that an operating system meets its security requirements as well as functional requirements is crucial. In this paper, we offer suggestions on how to test a secure operating system based on our testing experience on the Army Secure Operating System (ASOS). ASOS is a family of secure operating systems: the Dedicated Secure Army Secure Operating System (DS ASOS) designed for C2 level TCSEC [1] and the Multilevel Secure Army Secure Operating System (MLS ASOS) designed for the TCSEC A1 level. Both operating systems are designed for real time tactical applications coded in Ada and were developed using MILSTD 2167 [6]. This paper will concentrate on testing of Multilevel Secure ASOS.

1 Introduction

The goal of testing a secure operating system is to provide *assurance* through testing methodology that a system meets both the requirements detailed in the DoD Trusted Computer System Evaluation Criteria (TCSEC), otherwise known as the Orange Book, and its own specific functional requirements.

At a minimum, A1 security testing needs to concentrate on the security requirements found in the Orange Book. These security testing objectives are defined in section 4.1.3.2.1:

1. The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation.
2. A team of individuals who thoroughly understand the specific implementation of the TCB (Trusted Computing Base) shall subject its design documentation, source code, and object code to thorough analysis and testing.
3. Their objectives shall be: to uncover all design and implementation flaws that would permit a subject external to the TCB to read, change, or delete data normally denied under the mandatory or discretionary security policy enforced by the TCB; as well as to assure that no subject (without authorization to do so) is able to cause the TCB to enter a state such that it is unable to respond to communications initiated by other users.
4. The TCB shall be found resistant to penetration.
5. All discovered flaws shall be corrected and the TCB retested to demonstrate that they have been eliminated and that new flaws have not been introduced.
6. Testing shall demonstrate that the TCB implementation is consistent with the formal top-level specification.
7. No design flaws and no more than a few correctable implementation flaws may be found during testing and there shall be reasonable confidence that few remain. Manual or other mapping of the Formal Top-Level Specifications (FTLS) to the source code may form a basis for penetration testing.

These security objectives were used as initial guidelines for testing ASOS. Although necessary, they are not sufficient to provide full assurance. They deal solely with the Trusted Computing Base (kernel plus trusted software) and the security mechanisms. A secure system is more than just its TCB. It consists of a set of components that must function in an integrated manner. A system could be secure but not function properly or not be usable. Overly restrictive security mechanisms will certainly satisfy the requirements defined in the Orange Book but could make a system unusable. The Integration and Test team is in a unique position to assure that the system is usable because it is first user of the system.

In the paragraphs that follow, we present the testing methodology used on the ASOS project for its Multilevel Secure ASOS.

2 A New I&T Role

On ASOS, we found that the Integration and Test team needed to play a stronger role in the development of a secure system than the role usually defined for it in the development of a system. Usually, an I&T (Integration and Test) team is not in place at project inception. Its purpose is to test overall system requirements and/or test integration of units. In general, it has no voice in requirements or design and usually understands only portions of the system. The tests are usually built by a different part of the project (for example, tests are taken from development without much understanding of the test code) and test requirements only. Consequently, the testing of a system is limited to the scope and completeness of requirements definition. Therefore, it is possible to pass all of its requirements but not be usable or properly function. There would be no assurance that a particular security class has been met, least of all an A1 class. Whereas it is always important to have testing experience, knowledge of the totality of the product being tested is not at all necessary.

But for a secure system, as required on ASOS, security must be part of the design and test approach from the inception of the project, and it requires experienced developers and test engineers.

3 ASOS Testing Approach

The basic ASOS testing approach covers the following aspects:

1. Sound technical team.
2. Well planned methodology of testing
3. Need to integrate from beginning with all areas of project.
4. Explicit assurance from Orange Book.

3.1 Sound Technical Team

The development and test of a secure OS is a highly technical undertaking and requires a high level of technical expertise. Knowledge of operating system functionality and security are requisites for all members of the team. This includes the test engineers who can fulfill the security test objectives 2 and 3 in Section 1 only if they have a thorough understanding of the design and implementation of the system.

Additional support for a sound technical team can be found in the Orange Book. The guidelines on security testing state at least two of the test team members shall have previously completed a security test on another system. On ASOS, all the members of the I&T team had testing experience with the testing of the Dedicated Secure Operating System (C2 ASOS). In addition to the test experience from the Dedicated Secure Operating System, two members of the team had prior testing experience from the Interim ASOS Secure Operating System, a prototype of C2 ASOS. This testing experience gave the test team the necessary background in security and operating systems on which to base testing and, specifically, to design the high level security tests described in 3.4.

3.2 Well Planned Methodology of Testing

A well thought out plan of how testing will be done must be established early in the project. A coherent philosophy of testing must be established and documented in the test plan. This plan begins with how Unit testing, testing of the smallest, manageable entity of code, would be performed and documented and proceeds with a plan on how to integrate these units into subsystems and finally, the system. On ASOS, we established our plan for testing in the early development of DS ASOS and augmented it for A1 level security for MLS ASOS. We established, designed, coded, and tested our units as required by MILSTD 2167 and integrated them into subsystems as per the plan we defined in our Software Test Plan [4].

Most of our requirements were tested during Unit testing for early detection and correction of errors and retested during integration and system testing. Since operating systems are fairly complicated, we found it desirable to piece the operating system together incrementally. This provided an early view to the capabilities and allowed insight into the workings of the security mechanisms. It was essential that the access checking routines be tested early and any errors be corrected early. For example, the Integration and Test team found an error in the mandatory access checking routine which is used by many different routines in MLS ASOS. The Integration and Test team not only detected errors but also often recommended solutions. We were in a unique position to recommend solutions because we were familiar with the design as well as the requirements. Our incremental testing of both the DS ASOS and MLS ASOS allowed us to isolate and correct errors much earlier than if we had tested the system as whole.

An important philosophy for any system is to have repeatable tests, especially for a secure system. This includes unit tests; in fact, unit tests should be incorporated with integration tests to form a large test suite to check out the system when changes have been made. Repeatability becomes crucial with a secure system; it assures a high level of confidence by validating that the workings of security mechanisms have not been altered when a change has been made. On ASOS, we began generating a test suite for the DS ASOS and continued adding tests through its development. In some instances, the Integration and Test team worked with the Development team to develop test code (for example, file management). During the MLS ASOS development, we augmented these test cases for mandatory access checks, audit alarms and other specific A1 requirements generating a huge test suite.

Tests for an operating system should be self-checking, i.e., the test software checks if the criteria has been met and determines whether the test case passes or fails. For example, when testing the kernel task management read/write channel mechanism, the test software verifies that the appropriate message was received and no additional messages were received for the test to pass, else it fails. Self-checking tests not only allows repeatability but also allows for easy modification and generation of more test cases. On ASOS, we used procedures which allowed us to test approximately 88% of our requirements using one driver. Each time we needed to test new changes to the operating system, this driver would test the system, and we were assured that if it was successful, no errors that affected previous test cases were introduced into the system.

Each increment of software should include regression testing of the previous increment to ensure compatibility. This allows tracking of where and when errors are introduced into the system. On ASOS, each new increment went through regression testing before the new capabilities were tested, giving us confidence that basic capabilities were working before proceeding. For example, when the demand paging logic was added to memory management, all our previous tests were first run without demand paging being turned on to assure no breakage had been introduced. Testing of demand paging code followed upon successful completion of non-demand paging tests allowing us to isolate problems associated with demand paging.

Another important part of the ASOS methodology was the use of tools which aided in the performance and analysis of testing. On ASOS, a tool was developed to read and analyze audit data. It enabled us to demonstrate that all security auditing events had been generated. Another area where tools were very useful is in the area of human-machine interface testing. Traditionally, the human-machine interface testing has been difficult to repeat exactly and even if doable, the time frame was prohibitive. On ASOS, we created several tools which allowed us to automate feeding of input to terminals and capturing output from the terminals. This allowed for "exact" repeatability of some crucial areas of the operating system.

Thoroughness of testing is assured by mapping test cases to requirements. Each requirement must map to one or more test cases at one or more levels of testing (unit, integration, and system testing). On ASOS, requirements were mapped to levels and test cases in several iterations by the I&T team guaranteeing that each requirement was tested. We demonstrated 93% of the requirements during acceptance testing.

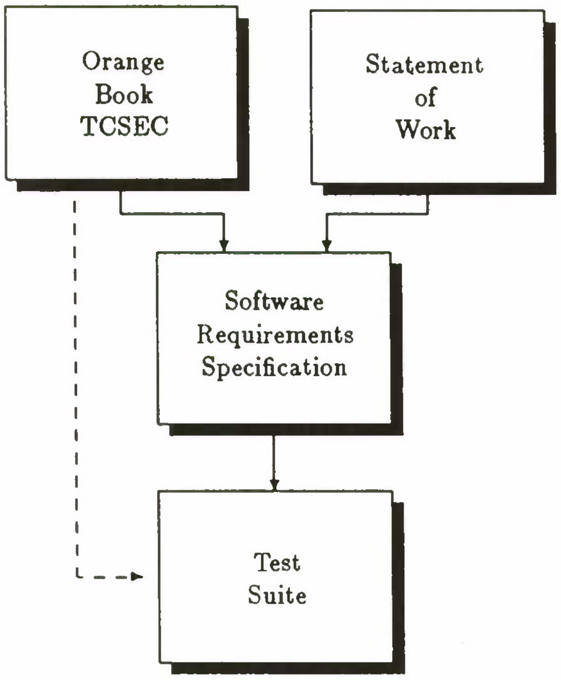


Figure 1: Requirements Derivation

3.3 Project Integration

In addition to its role of designing tests, documenting tests in the Software Test Description document and procedures for performing these tests in the Software Test Procedure Document, and performing integration and system tests, the Integration Test team must be part of each phase of the project. I&T's role in a secure system begins at project inception with the requirements definition phase and ends with acceptance testing and delivery. It is the responsibility of the I&T team to demonstrate that the operating system meets both its security and functional requirements and is usable. On ASOS, this goal was achieved by integrating I&T's influence in all aspects of the project.

ASOS began with an I&T team in place at the start of the project and I&T was an integral part of each phase of the project. This included phases that a traditional I&T would never have been a part of such as:

1. Requirements definition
2. Verification
3. Design
4. Development
5. Configuration Management

In the paragraphs that follow, we elaborate on the role the Integration and Test team played on the ASOS project to help reach the goal of a secure, usable, and well tested Multilevel Secure ASOS system through its participation in the above phases of the project.

3.3.1 Requirements Definition

During the Requirements Definition phase, the requirements for the operating system are written and documented in the Software Requirements Specification (SRS). On ASOS, the requirements were derived from the Orange Book and the Statement of Work (SOW) as illustrated in Figure 1. The SRS was written according to the standards defined by MILSTD 2167. These requirements define the system and are used as the basis for both developing and testing the system. Each requirement must be testable. The system is finally

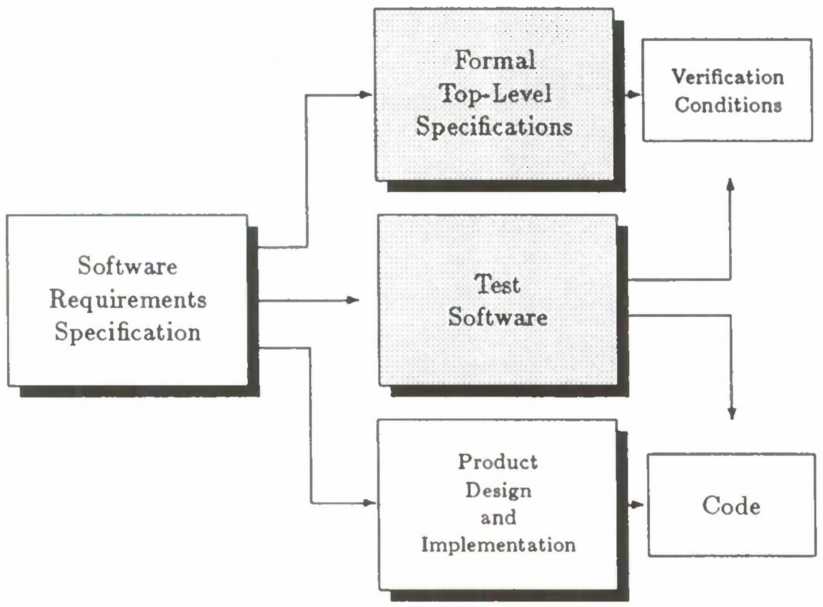

Formal Top-Level Specifications
Verification Conditions
Software Requirements Specification
Test Software
Product Design and Implementation
Code

Figure 2: FTLS and Test Software Relationship

accepted on the basis of whether these requirements are satisfied. As part of the requirements definition, a mapping of test requirements to test levels (unit, integration, and/or system) is generated to ensure that each requirement is tested at least once.

On ASOS, I&T played a vital role in the requirements definition phase. Not only did I&T determine whether or not a requirement was testable and at what level it should be tested, but I&T was also responsible for guaranteeing that the requirements in the Orange Book were represented in the SRS. In a secure system, each Orange Book requirement must map to one or more SRS requirements. I&T had the job of highlighting security specific requirements in its test suite. A demo of the human-machine interface highlighting security requirements was developed and incorporated as part of the System Test. In the MLS ASOS, this meant highlighting mandatory access checks and audit alarms as well as discretionary access checks and auditing. I&T determined which requirements needed to be added, deleted, or reworded based on the Orange Book, SOW, and testability. The team also determined which requirements could be shown at the system level and which requirements had to be demonstrated at the unit level testing. For example, if testing of a requirement necessitated looking at an internal data structure which is outside the scope of integration, then it was allocated to the unit level.

3.3.2 Verification

One of the key requirements of security testing (objective 6 in section 1) is to demonstrate that the TCB implementation is consistent with the Formal Top-level Specification (FTLS). This consistency is an aspect of testing not found in traditional development environments. It has application for those systems whose design has been specified in a formal language and rigorously proved to maintain a security policy.

The reason this consistency is important is that it is required to bring the entire effort of formal verification to fruition. Without this step it is very possible to expend a great effort to specify a design, prove it is secure and then implement a product that only remotely resembles its proven design. For this reason, the ASOS I&T test team has been involved directly in developing the ASOS security model, engineering the verification plan, and developing the FTLS.

In TRW's approach to the development of a secure operating system, there is a strong correlation between the test software and the formal specifications. This is because both are derived from the same requirements document (as is, of course, the developed code) as shown in Figure 2.

An automatic generation of test cases from the FTLS would be an ideal approach and research should continue along this avenue. A TRW study to this end has been conducted, the RADC (Rome Air Development Center) Computer/Communication Security Study (RCCSS) (see [7]). For now, only a "consistency" is required.

At a minimum this consistency would mean that security testing must demonstrate that the TCB (both Security Kernel and Trusted Software) correctly corresponds to its specification. This level of "consistency" is generally tested during the kernel interface testing and nominal trusted software functional tests.

The ASOS I&T team felt that a stronger degree of assurance would result from directly testing the actual verification conditions derived from the FTLS as shown in Figure 2. This form of testing was delegated to unit testing (see 3.2) demonstrating that the necessary state variables are invariant for each thread when an exception condition is tested, and that they reflect the correct state transition otherwise.

3.3.3 Design Phase

The basic foundation of any development is its requirements. From these, design and implementation follow along well known paths with the intent of complying with the requirements. Testing, in its basic sense, needs to look very closely at requirements. A statement regarding the failure of a product to uphold a design or implementation approach is not as strong as one that regards a requirement not being met. This was discussed in section 3.3.1.

Though requirements are the basic driving force behind both implementation and testing, it behooves a project to ensure that a particular design approach, well known and documented, finds its way into the product. This is especially the case when requirements are written at a somewhat high level. An analysis and test of the design and implementation is explicitly called for (test objective 2 in section 1).

Indeed, there may be little material in a requirements document for a test case developer to use in devising how to test a particular requirement. Knowledge of a design approach may give the test engineer alternate approaches or test cases where the design is really exercised.

For example, often times a demand paging requirement is simply stated, leaving freedom in the design. This is proper, and a basic test approach would be to execute programs that exceed the physical memory. Knowing whether object code is shared or the criteria for choosing a page for overlay would influence whether multiple instances of one program or multiple programs are chosen for a test approach.

Insight into the design was utilized by one of our basic security mechanism tests (see section 3.4, MAC_CHECK) during which the ASOS kernel routine regarding the basic dominance mediation between security levels was tested with many permutations. Knowledge of the design and implementation of the functions allowed our test engineers to devise *alternate* functions that perform each check in parallel to check the correctness of the result.

3.3.4 Development Phase

Any type of testing effort is beneficial early and throughout the development phase. This just makes good sense as the earlier a problem can be found and rectified the better. In this way, the mistake of building upon bad early code and becoming dependent on its erroneous behavior can be avoided as well as the cost of correction.

In the case of security, catching an errant routine early can avoid disaster. The heart of the TCB is the security kernel and the heart of the security kernel is set of routines that perform security mediation. If these routines are not rigorously tested early (see test objective 3 in section 1) and the operating system's security is based on them, then any subsequent error could have a rippling effect through the system with severe cost and schedule impact.

Involvement of the test team with the early ASOS kernel increments uncovered an error in the mandatory security checking routines that enabled us to right ourselves and avoid such a cost. A routine was developed to return a boolean for

$$dominates(a, b)$$

where a and b are security levels and the function returns true if a dominates b. This routine was implemented correctly; however, a routine to perform a "strictly" dominates function (where a dominates b and they are unequal) was coded in terms of *dominates* and an idea of total ordering (as found in numbers) such that

$$s\_dominates(a, b) \equiv \neg\, dominates(b, a)$$

since, with numbers,

$$a > b \equiv \neg (b \geq a).$$

The test cases chosen by the developer did not involve disjoint category sets and all his tests passed. The I&T test team found early that if a and b had no dominance relationship, *s_dominates* would erroneously return true.

The advantages of finding this kind of error at the threshold of building an entire TCB around such routines cannot be overstated in terms of cost savings of fixing the error early. Not to be overlooked is the advantage of an independent test approach as illustrated here. This gaping hole in security may not have been found without the additional set of test cases.

Also, though not strictly a security issue, the correctness of the tested product is going to depend ultimately on the correctness of the test software which has the undesirable possibility of not finding bugs as well as reporting false bugs. Being able to "exercise" test software early on the developing product makes for stronger assurance statements.

3.3.5 Configuration Management

Most testing efforts involve integration of components into a whole system. In the case of the original development of a system, the only sensible way to accomplish this integration is to start with the nucleus of the system, integrate it, test it, and expand upon this tested baseline repeating the cycle until the whole product is integrated and tested. The ASOS I&T team devised a series of logical steps in the development of the system starting early with the nucleus of the security kernel and adding capabilities in software builds.

This incremental build activity must be supported by a configuration management mechanism and tool set. The testing organization should have the most influence in this area as they are most affected. Configuration management must be utilized for not only the design and development but for the life-cycle of the product.

This concept is especially appropriate for a secure system as the configuration management system is also responsible for assuring that only authorized updates of both software and hardware are made to the system. Not only must configuration management have capabilities to control changes, but it must allow for automatic regression testing upon any changes.

Under ASOS, the I&T organization monitored all build activities in which the ASOS TCB and test software were rigidly controlled. Any change resulted in the new baseline being subjected to a full suite of regression tests. Though somewhat tedious, this procedure was faithfully adhered to, for good reason: In a complex system, a change is never really isolated. It was always amazing to see how a seemingly innocuous change in one part of the system could manifest some grave problems in another, and how, if only the portion of the test suite involved in the area that was changed was run, the error would not have been discovered as soon.

One example comes to mind: The object reuse requirement led us to adopt an initialization design calling for the clearing of each data segment not allocated to an existing file. This design was implemented by an uninterrupted I/O loop in the kernel (after all, no application is running during initialization). The initialization test suite was rerun, of course, and showed all tests continuing to pass. The ASOS I&T testing philosophy, however, called for running *all* tests, and we found right away, that in the file management system call code, when a disk was mounted, the same code was used to clear the unused disk segments. Of course, the applications on the system stalled during the loop, and we realized an interruptible design was called for.

3.4 Explicit Assurance from Orange Book

As shown in Figure 1, the requirements document for a secure system should be derived, in part, from the requirements for the proper class from the Orange Book.

In the development of ASOS, Orange Book A1 requirements map into the system requirements. Thus, the testing of these requirements will, in effect, test the systems adherence to the requirements for an A1 system.

The ASOS test organization felt that a stronger statement can be made that the system is A1 if Orange Book requirements are explicitly tested by a unique security test case designed for that purpose.

A major ASOS security testing document [5] describes a full set of security related test cases explicitly mapping several test cases to each section of the A1 criteria in the Orange Book.

The requirements areas found in the Orange Book are listed below along with the test cases found in the ASOS Security Test Case Description document that pertain to that area. The name in bold-face is the actual test case name used ([5]).

1. **Penetration** - test objective 4 in section 1.

   Parameter Tests check the ability of the kernel software to detect and properly report as a parameter_error any input parameter whose value is outside the constraints imposed by the kernel for that particular parameter's type. This method tests the integrity of the kernel. Since the kernel is separately linked, it is possible to circumvent Ada type checking during the domain switch to the kernel. Parameter values outside the type constraints may also be specified by calling the kernel directly by TRAP instructions from assembler programs. Several serious security problems could result if the kernel used an unchecked input parameter including:

   - Invalid areas of TCB data could be accessed.
   - A fatal memory fault could occur.
   - A fatal divide_by_zero fault could occur.

   The parameters under consideration in this test are scalar input parameters assumed to be passed by value.

   Parameter tests also check the ability of the kernel software to detect and properly report as a parameter_error any parameter whose address value is outside the program instance's virtual address space. If such parameters are unchecked, the kernel may deliver output data into its own address space or that of another program instance. It may also encounter a fatal memory fault. Parameters under consideration for this test are those passed by reference where it is assumed that an address is passed.

   Kernel_Entry verifies that random kernel calls passing random parameters have no adverse effect on the kernel's processing.

   Kernel_Consistency checks the susceptibility of the security kernel to regulated kernel state sequences. This test focuses on kernel calls originating nominally from nonkernel ASOS and expected in a nominal sequence. It determines whether the kernel conditions itself to expect a certain cadence of kernel calls. An example might be two successive calls to resume the same task using the Task_Resume kernel call. This kind of call is normally used by the Ada RSL and would not be called twice successively for the same task. Regardless, the kernel should be able to withstand this kind of anomaly.

   Secure_Recovery determines whether, upon recovery, the TCB successfully places itself in a secure state. One example from this test case follows the scenario:

   (a) Create files writing a pattern of data to the disk until the disk capacity is exceeded and then delete these files.

   (b) During the file deletion phase, prompt the user at a random time to hit Carriage Return.

   (c) Once the Carriage Return is entered, call terminate system with the restart option.

   (d) Examine all available free disk pages and look for any uncleared data which exists.

   (e) If any uncleared data is detected, report the anomaly.

   This method tests a crucial security requirement regarding object reuse. When a file is deleted, there are timing windows that must be considered in a secure design, if there is a possibility the system could go down. When a file is deleted, all segments used by the file are made available for reuse. They also must be cleared. If these two objectives are simply followed as stated then, if the system goes down during the clearing phase, several disk segments may be both available and *uncleared.* The ASOS design calls for clearing first then making available each segment in turn. If a crash occurs, there may be uncleared segments, but they could not be used again. During initialization, these segments are found (since no file system directory owns them) and they are cleared and made available.

So, this test verifies that the system is restarted in a secure state:

- All kernel initialization checks pass.
- File Management checks for data structure consistency pass.
- All available disk pages are clear.

2. **MAC policy** - test objective 1 in section 1.

   This test case is designed to test the kernel's Mandatory Access Control (MAC) routine. The ASOS access level consists of a security classification (1-16), an integrity classification (1-16) and up to 64 security categories and 64 integrity categories. The number of different access level comparisons that the kernel's MAC routine can perform is therefore too large to test with all permutations. A representative sample of the subject and object access level combinations are tested. These access levels are chosen randomly. The scenario for testing is:

   - Verification of the implementation of the mandatory access policy is performed by interfacing directly to the kernel's MAC function.
   - Access levels are composed of Security *and* Integrity components.
   - The Kernel's MAC function determines whether mandatory access is allowed based on:
     - Desired access (Read, Read/Append, etc.)
     - Subject's access level
     - Subject's Privileges
     - Object's access level.
   - The test program selects random input values from a full range of classifications (16) and categories (64).
   - The results are verified by an independent checker routine as described in section 3.3.3.

3. **DAC policy** - test objective 1 in section 1.

   This test case is designed to test the kernel's Discretionary Access Control (DAC) routine. The test scenario verifies discretionary access policy is implemented correctly by interfacing directly to the kernel's DAC function.

   In particular, the kernel's DAC function determines whether discretionary access is allowed based on:

   - Desired access (read, read/append, etc.)
   - Subject's privileges
   - User's access rights to the object
   - Group's access rights to the object

   The results are verified by an independent checker routine which, like the routine to check the MAC function, independently concludes whether a requested access is allowed and checks to see if the function being tested arrived at the same conclusion.

4. **Object Reuse**

   File_Object_Reuse checks the ability of the TCB to clear the contents of disk space before granting access to subjects.

   Channel_Object_Reuse checks the ability of the TCB to restrict the reading of extraneous data through the channel mechanism.

   Register_Reuse verifies that the data registers are cleared prior to kernel task management allowing a new task to run. Since the new task to run may or may not be in the same program instance as the current task, the data registers must be cleared by the kernel between task switches in order to prevent residual information from passing from one task to another.

   Device_Object_Reuse checks the ability of the TCB to initialize the device registers before granting access to subjects. It verifies that device space is cleared when a device is unmapped.

   Memory_Object_Reuse checks the ability of the TCB to clear the contents of memory segments before granting access to subjects.

5. **Denial of Service** - test objective 3 in section 1.

   Time_Slicing tests the kernel's ability to perform time slicing between two tasks of equal priority under the condition that the actions of task1 will cause a momentary preemption of time-slicing between the two tasks by a higher priority task.

   - A test program activates two tasks of equal priority:
     - Task 1 repeatedly performs auditable actions.
     - Task 2 performs actions which are not auditable.
     - The Secure Audit Capture preempts the two tasks when handling Task 1's audit events.
   - This test verifies that task 1 and task 2 are served equitably by the processor.

   This test case attempts to provide a scenario where two tasks of equal priority are both considered for scheduling simultaneously and checks that, over time, they receive the same amount of CPU time. For this test, they both must be uninterruptible by nature and still be rescheduled. The Secure Audit Capture program, which runs at the highest priority, provided the preempting agent.

   When this test case was first run, a bug in the Kernel resulted in Task 1 locking out Task 2. This test case was instrumental in fixing this error.

   File_System_Denial checks to see whether a low priority program can deadlock or effectively disrupt the File System to other users. The test program attempts to overload the File System by progressively activating a low priority task which repeatedly requests File Management Services. The test program performs five iterations. Each iteration places a heavier load on the File System than the previous iteration. For each iteration, a high priority task (which also requests File Mangement Services) is activated as a "sample" task and its completion time is measured. Any variances in the completion time of the sample tasks in relation to each other is reported.

6. **Architecture**

   Executable_File_Protection checks the TCB's ability to restrict all attempts at modifying the program executables, even if the user has the proper access controls and privileges. This construct has been placed in the system architecture in order to effectively block the propagation of computer viruses in ASOS.

   - Verifies that program executable files cannot be deleted or appended to
   - Checks ASOS virus protection mechanism

   Access_Illegal_Memory tests the kernel's ability to restrict a program's attempt to access portions of the programs virtual memory space illegally.

   The test program attempts to access data illegally:

   - by attempting to read and write data to locations outside of the legal virtual memory space of the program.
   - by attempting to write data to read only portions of the programs virtual memory space.

   Interactive_Security_Demo demonstrates the various security features of the Multilevel Secure Operating System via the Secure Server, System Administrator, and System Operator functions. This is an interactive test.

7. **Devices**

   Allocate_Invalid_Device checks the ability of the map_io kernel call to restrict device mapping to legitimate user devices. A user should not be able to map a device that is reserved as a system device or one which was not declared at system generation.

   User_Printer checks the restriction placed on the use of the system line printer by the TCB. The TCB allows a program to map the printer device only if the program possesses the proper mandatory and discretionary access to the device. Discretionary access to the printer device is initially set to no access for all users.

Modify_Print_Buffer attempts to modify the print buffer prior to the actual printing of the file.

A low access program attempts to access files copied into the spool directory by the Line Printer Spooler as the file is printed.

Spooler_Spoofer determines whether the line printer spooler/despooler (the trusted program that receives print requests, queues them, and drives the printer) is able to validate and control requests reserved for the operator.

Masquerade_Print_Label attempts to deceive the system operator into thinking that a particular print request is of a lower security classification than it actually is by attempting to violate the Operator's trusted path to the printer.

8. **Access Controls**

   These tests determine whether an unprivileged user without the proper discretionary and mandatory access rights to the authentication database or audit trail is able to access them.

   - **Authentication_Database_Read**
   - **Authentication_Database_Write**
   - **Audit_Trail_Read**
   - **Audit_Trail_Write**

9. **Auditing**

   Jumble_Audit_Data checks the system's behavior when attempts are made to deceive the System Administrator via the Audit Display mechanism.

   Audit_Data_Loss checks the TCB's ability to capture all auditable events under a simulated duress condition. This test involves repeated requests to create new audit files to determine whether the TCB can continue to capture all audit events.

   Audit_Space_Limit checks the system's behavior when there is no more disk space available and auditing is active. This test verifies:

   - System is terminated by Secure Audit Capture.
   - System can be re-booted and audit files accessed.
   - Minimal amount of audit data loss occurred.

   Generate_All_Audit_Events verifies that all defined auditable events are audited by the system and that the audit events are then written to the audit file by the Secure Audit Capture trusted program.

10. **Covert Channels**

    - Verifies a covert_channel_usage audit event is generated every time a covert channel is reported.
    - Verifies an audit_alarm audit event is generated whenever a program's bandwidth has exceeded the defined threshold and rate for the covert channel being tested.
    - Verifies that a program is delayed when the bandwidth exceeds the allowable limit.

3.5 Flaw Hypothesis Methodology

Many of the security tests defined in Section 3.4 especially those dealing with penetration, were devised using the Flaw Hypothesis methodology, which is used to infer possible weaknesses in an operating system and requires detailed knowledge of the system architecture and design.

First potential flaws were "proposed". Some flaws were precluded by the MLS ASOS System design (e.g., symbolic links, executable search paths). Other flaws were not precluded by design, but were ineffective (e.g., bogus LOGIN routines are rendered ineffective since the system times out after a certain number). Remaining flaws were tested, where feasible, and confirmed or rejected.

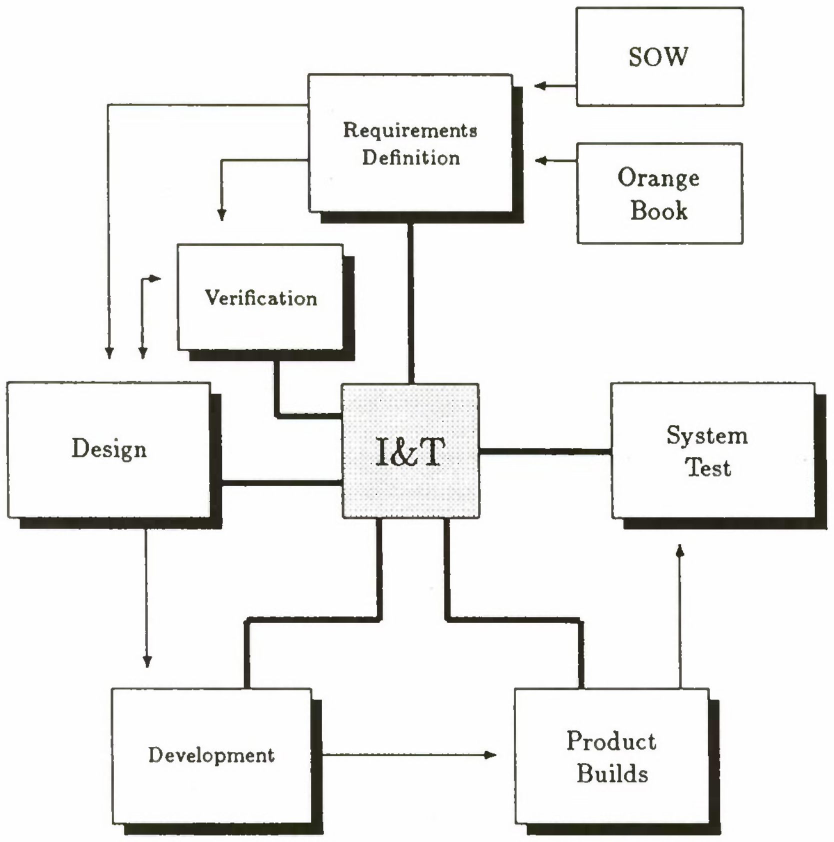


Figure 3: Integration & Test: An Integral Part

If a flaw is confirmed, a software problem report is written. When fixed, tests are rerun to verify that the fix corrects the problem. In some cases, once a flaw is corrected, the new design renders testing unnecessary. If regression testing of the fixed flaw is desired, the test is incorporated into the Security Test suite.

One example of a security test case that existed to uncover a flaw and then was precluded by a changed design was regarding the generation of program ids:

The security kernel must ensure that the generation of ids associated with new program instances (processes) are not predictable. If they were, then program instances (security subjects in ASOS) could communicate over a covet channel by the receiving subject noting whether a program instance it just activated got the next id or the one after it. The difference would signal whether the sending subject activated a program and thus a binary message could be encoded.

The ASOS I&T team proposed this flaw, checked the kernel code, and saw that each new id was obtained via a function based on current numbers of tasks and programs in the system. It was possible to predict each new id and a test case was written and run to prove this prediction was possible and to substantiate a Software Problem Report. The flaw was subsequently fixed with a completely new design rendering the test case, though crucial in its day, unnecessary to run thereafter.

It is through this cycle of test, report, fix, and retest that assurance is gained for test objectives 5 and 7 in section 1.

4 Summary

Our experience with testing Multilevel Secure ASOS has shown us that the Integration and Test team needs to have a strong role in the engineering of an operating system designed for A1 to ensure that the security and functional requirements are met and that the system functions properly. Based on our experience on ASOS, we recommend the following:

- Due to the additional requirements that security imposes, the I&T team must be of a high caliber of technical expertise, versed in security fundamentals, and have previous test experience.
- The method of testing must be well thought out and allow for early visibility into the security mechanisms.
- Test cases must be repeatable and self-checking whenever possible.
- I&T must be an integral part of all phases of the project from project inception as illustrated in Figure 3 and explained throughout this paper.
- Security testing must go beyond normal requirements testing and into areas such as penetration, denial of service, and specialized architecture tests.

5 Acknowledgements

The ASOS development is funded by the U.S. Army Communications-Electronics Command and the National Computer Security Center under Contract No. DAAB07-86-CA032. The authors would like to acknowledge Bruce Beaumont, Ann Engvall, and Frank Esteves who were a vital part of the ASOS Integration and Test team and Eric Anderson who manages ASOS. We would also like to thank Eric Anderson, Bruce Beaumont, Cristi Garvey, Ruth Hart, and Jim Ramsay for their constructive reviews of this paper.

References

[1] National Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, 1985.

[2] Army Secure Operating System (ASOS) Multilevel Secure Operating System Software Requirements Specification, CDRL H021 under contract DAAB07-86-CA032 (TRW Defense Systems Group), 1985.

[3] System Security/Verification Plan for the Army Secure Operating System (ASOS) CDRL F001 under contract DAAB07-86-CA032 (TRW Defense Systems Group), 1987.

[4] Multilevel Army Secure Operating System (ASOS) Secure Operating System Security Software Test Plan, CDRL H031 under contract DAAB07-86-CA032 (TRW Defense Systems Group), 1989.

[5] Multilevel Army Secure Operating System (ASOS) Secure Operating System Security Software Test Description, CDRL H033 under contract DAAB07-86-CA032 (TRW Defense Systems Group), 1989.

[6] DOD-STD-2167, Defense System Software Development, 1985.

[7] Rome Air Development Center Computer Communication Security Study, Final Technical Report "Task 2: Advanced Test Methodology", 1988.

An Assertion Mapping Approach to Software Test Design

Greg Bullough
Jim Loomis
Peter Weiss

Amdahl Corporation
P.O. Box 3470 (M/S 214)
Sunnyvale, California, 94088-3470

ABSTRACT

A test design method was developed for security features of a B2-level secure operating system. The method consists of dividing requirements and design documents written in plain English into logical "assertions." The assertions are then mapped between successive levels of design abstraction. The mapping process checks the consistency between requirements and design, identifying errors at early stages of development. Classical test techniques supplement the assertion-mapping approach to enhance test coverage. The test design process provides a documentation trail for all requirements and design elements, from the TCSEC through the final design of the system features and resulting test cases.

1. Introduction

In an ideal world, engineering of a complete project would always proceed in an orderly fashion. It would move through increasingly detailed levels of abstraction, using formal specification languages at each level. The task of validating such systems, while very detailed, is straightforward; the mechanical nature of the specification languages clearly identifies the items to be tested.

In the "real world" however, software systems frequently evolve from earlier versions of themselves. As such, the existing system will probably have little or no basis in formal software engineering methods. Even the most rudimentary of design documents are often absent. Where design documents exist, they frequently consist of English text descriptions of the various sub-functions.

Even in the case of new software development, the use of formal specification languages is the exception, rather than the rule. The use of informal methods remains as the dominant means of expressing the parameters of software behavior.

All of this may present a problem for the test team which is attempting to develop validation suites for a secure computing system. They frequently have little or no control over the format of the design documents. It is up to them to devise a strategy to utilize the design documents as they are given.

Enhanced security features are being implemented in a new version of UNIX System V® to be submitted for evaluation by the NCSC. The target TCSEC[1] security level is B2. This paper discusses an informal method (as distinguished from formal methods) used for functional (i.e., 'black-box') testing of security features in support of the B2 evaluation.

The test team has several goals, both in the area of testing and in the more general realm of "assurance:"

- To ensure that the design of the software is sufficient to meet the goals of the product (e.g. to achieve an NCSC B2 rating).
- To validate that the software behaves as designed.
- To be able to state, with reasonable certainty, that the functions under test have been adequately covered by the testing process.

To meet these goals, the test-team developed an integrated test method which will be discussed below.

2. *Assertion Mapping*

Although natural language was used to specify the system design, it was believed that a structured method of performing informal verification was both necessary and achievable. The approach consists of dividing the requirements and design texts into logical "assertions," and then mapping the assertions between successive levels of design abstraction.

2.1 *The Nature of Assertions*

For our purposes, an assertion is defined as a single, simple, complete and verifiable logical proposition. For example, the following is a hypothetical assertion derived from the discretionary access control feature requirements:

> The *setacl* command will support the changing of discretionary permission information associated with an object.

is an assertion. However,

> The *setacl* command will support the removal and display of discretionary permission information associated with an object.

needs to be simplified into two assertions, thusly:

> The *setacl* command will support the removal... of discretionary permission information associated with an object.
>
> The *setacl* command will support the ... display of discretionary permission information associated with an object.

On the other hand, a statement such as

> The *setacl* command provides a consistent interface to the dac mechanism.

is not an assertion. It is instead a general statement about a design feature. The statement is not specific enough to be testable. In a plain-English design document such statements serve explanatory and linkage functions. The elimination of such statements from assertion lists requires a certain degree of judgement on the part of the test designer.

2.2 *Source Documents*

There were three types of source documents which were used to generate assertions on this particular project. They were:

1. TCSEC - The Trusted Computing System Evaluation Criteria or "Orange Book," published by the Defense Department. Since the motivation for development of the particular system was to obtain a B2 rating, this document represented the primary standard for the assurance process.
2. Requirements - A requirements document was prepared which contained external specifications for the UNIX System V® features which would be developed to meet B2 requirements. This included syntax and semantics of new commands and system calls, as well as more general requirements.
3. Design - Each *feature* (a functional subdivision) of the system is documented at this level. Design documents describe internal and external functions in detail. They usually include implementation details which are beyond the scope of the requirements document. Each individual design document corresponds to a sub-section of the requirements document.

2.3 Assertion List Development

Each source document serves as the basis for a distinct "assertion list." The assertion lists are developed for each security feature area (e.g., trusted path) from the source documents.

The process of developing assertions consists of surveying the source documents sequentially and extracting textual fragments which comprise assertions. As far as possible, direct quotes are used. Standard quoting rules are followed. Text added for clarification is included in square brackets ([...]), and areas where text is deleted are denoted by the ... symbol. (See the example of a divided complex assertion in a previous section.)

Assertions are named according to their feature, component, source document, and sequence number. For example, "dac-setacl-R1" represents the first assertion from the section of the requirements document concerned with the "setacl" component of the "dac" (discretionary access control) feature. The sequence numbers of design-document assertions are designated by "D" (e.g. "dac-setacl-D1).

Once identified in a source document for a feature, assertions are placed into lists by component. They are represented in the following format:

dac-setacl-R1	(5.4.1) The *setacl* command will support the changing of discretionary permission information associated with an object.
dac-setacl-R2	(5.4.1) *setacl* can be used to set an ACL by the owner of a file

Optionally, a section number from the source document may be included following the assertion name, for ease of reference at a later time. The organization of the assertion lists by feature and subcomponent categories will simplify the mapping process which is to follow.

2.4 The Mapping Process

The mapping process performs an assurance role for the integrity of the development cycle. It provides a more structured review of the elements of successive design refinements than the more common peer review and walk-through. The process of mapping will often detect those elements of requirements which are erroneously omitted or contradicted in later design phases.

Assertion lists for each source document are mapped one to another between successive levels of design abstraction for each security feature (e.g., trusted path). In general, TCSEC assertions applicable to Class B2 are mapped to requirements assertions, which in turn should map to design assertions. Each mapped set of requirements and design assertions forms a "test element."

2.5 Mapping Rules

2.5.1 TCSEC to Requirements Mapping Rules

- Every assertion from the TCSEC must be mapped to one or more assertions in the requirements document which, taken together, are sufficient to meet the TCSEC requirement.
- Requirements may exist that do not map back to the TCSEC.
- No requirements document statement may contradict a TCSEC statement.

2.5.2 Requirements to Design Mapping Rules

- Every assertion from the requirements document must be mapped to one or more assertions in the design document.
- No design document statement may contradict a requirements assertion.
- No design document assertion may contradict a TCSEC requirement.

2.5.3 Design to Test Element Mapping Rules

- Each requirements assertion that is mapped to a design document assertion shall be identified as a discrete test element (e.g., TE0201).
- Any design document statement that cannot be mapped to a requirements document assertion shall be identified as a discrete test element.

2.5.4 Example of Mapping Rules Application

The following example illustrates the mapping process. A Discretionary Access Control (DAC) TCSEC requirement is mapped to the product requirements and design documents to verify that each TCSEC requirement is met. Section 2.6 will show how the output of the mapping process is further defined as test elements, which then become the building blocks for test cases. This process verifies that the design of the software meets the TCSEC requirements and leads to the creation of the test cases necessary to validate that the product functions as designed.

The TCSEC B2 level DAC requirements state:

> Access permission to an object by users not already possessing access permission shall only be assigned by authorized users.

By analyzing this statement we can derive:

dac-setacl-T1	a mechanism is needed to grant access permission to an object to users not possessing access permission

and

dac-setacl-T2	access permission may only be assigned by authorized users.

We can refer to these requirements as T1 and T2. The product requirements document must be shown to meet each of these TCSEC requirements.

The requirements document states:

dac-setacl-R1	The *setacl* command will support the changing of discretionary permission information associated with an object.

This assertion identifies the mechanism by which access permission to objects will be granted and so satisfies the first TCSEC requirement (T1).

The requirements state:

> *setacl* will allow the file owner or a process with appropriate privilege to set an ACL.

This statement can be be broken down into two assertions:

dac-setacl-R2	*setacl* can be used to set an ACL by the owner of a file
dac-setacl-R3	*setacl* can be used to set an ACL by a process with appropriate privilege

Together, R2 and R3 address T2 in that they state the basis for granting access permissions to an object.

The requirements document further states:

dac-setacl-R4	if the DAC check fails when a request is made to modify the ACL, then permission to modify the ACL will be denied

This assertion also addresses the T2 requirement. Thus R1 through R4 meet the TCSEC requirement.

The product design document shows how the requirements will be implemented. The design document states:

dac-setacl-D1	the *setacl -s* option will set an object's ACL
dac-setacl-D2	the *-a* option will add an ACL
dac-setacl-D3	the *-m* option will modify an ACL
dac-setacl-D4	the *-d* option will delete an ACL

D1 through D4 can be mapped to R1 as they enumerate the ways in which discretionary access permission may be changed.

The design document states:

dac-setacl-D5	*setacl* can be executed by a process with an effective uid equal to the owner of the object
dac-setacl-D6	*setacl* can be executed by a process with the appropriate privilege (P_OWNER)
dac-setacl-D7	If neither the process' effective uid is equal to the owner of the object nor the process holds the appropriate privilege (P_OWNER), *setacl* will fail.
dac-setacl-D8	If *setacl* fails either because the effective uid of the process is different than the owner of the object and the process does not hold P_OWNER, the error NOPERMIT will be returned.

D5 maps to R2. D6 maps to R3. D7 and D8 map to R4. That the command will fail and that the correct error message is returned are separate assertions and both must be tested.

From this it is apparent to what extent the design fulfills the requirements and the requirements fulfills the TCSEC objectives. Two TCSEC assertions have resulted in eight design assertions (see Figures 1 and 2). Note, also, that one requirements assertion, **dac-setacl-R5**, has no corresponding design assertion. Such cases will be flagged as errors, and require updates at least to the design documents to correct them. The stand-alone design assertion, **dac-secacl-D9**, is allowable as long as it does not conflict with the requirements.

The assertion mapping results in an appendix to each TDS (see Figure 2). The assertions are next broken down into test elements from which the test cases are built.

2.6 Test Elements

2.6.1 Test Element Composition

Each relation where some number of design assertions are mapped to a requirements assertion represents a unified logical implication. Such an implication might be stated for the general case, as "if the design assertions are true, then the requirements assertion is true."

Such a set of assertions is described as a "Test Element," and provides the basis for "white box" testing. It also provides a convenient way of denoting the relation of design assertions to requirements assertions.

In identification of test cases, test elements are used to determine what tests are necessary to achieve coverage. Thus, in the scheme under study, test elements, rather than assertions, are assigned to specific test cases.

It is required that each test element be covered within a test case. Related test elements may be grouped together for testing in a single case.

2.6.2 Division of Test Elements

It was found to be necessary to permit the division of test elements into subelements for a number of reasons:

- Permitting test elements to be tested across several test cases sometimes allowed more economy in the size and number of cases.
- Some test elements applied to a number of mutually-exclusive execution environments. Therefore it would have been difficult to test them in a single test-case, or indeed even in the same test suite.
- Dividing test elements into sub-elements permitted tracking of specific test inputs towards complete coverage of a test element. For example, valid and invalid input classes may be independently tracked, along with upper and lower boundary conditions.

The benefits (and indeed necessity) of permitting sub-division of test elements were deemed to be more important than the simplicity that would be retained by keeping them atomic.

Test elements which are subdivided are recorded in a tabular fashion, along with the circumstances which distinguish their sub-elements from one another. Sub-elements are numbered with the test-element number and subelement number (e.g TE215-3).

3. Test Cases

The test elements and test sub-elements are logically grouped together into "Test Cases." The criteria for grouping a set of test elements and subelements together are informal and primarily utilitarian. For example, error conditions for a particular command will frequently be tested in a single test case.

The "Test Cases" are named according to feature and component, and numbered in ascending sequence, (e.g., **dac-setacl-010**). The example is the first test case for the setacl component of the dac (discretionary access control) feature. The sequence numbering system uses an increment of '010' to allow for subsequent insertion of test cases as needed.

The test cases for each feature are contained in a Test Case Specification (TCS) document which is a superset of the *IEEE Standard for Software Test Documentation ANSI/IEEE 829-1983*. [2]

A TCS document specifies the following items for each case which is identified:

1. Description
2. Inputs
3. Outputs
4. Procedural Requirements
5. Test Elements

Parallel sets of input and output specifications cover the test elements included in the test case. The TCS provides complete specifications for coding a test case.

The test case specifications are stored in a database for each feature. A subset of the information, test case identifiers and descriptions are automatically extracted from the database for creating the Test Design Specification (TDS) document. An automatic tool scans the database to verify that the test cases cover all test elements and subelements for all components of a feature.

4. Integration of 'Classical' test methods

Test elements provide a documentation-based foundation for the application of classical software test methods, such as those described by Myers[3]. It is more usual for such methods to be applied directly to the design documents. By using the test elements as a basis, the tester can select test elements with the assurance that he or she is covering the full documentation base. The test elements unite the related assertions across the various documents, leading to a far more orderly development of test input specifications.

Such techniques as input class mapping and boundary analysis are employed in the selection of specific inputs during the specification of test cases.

For example, in the specification of TCS for a test case which covers TE0202, it will be recognized that it is a *valid* input condition which is being tested. That is, the input conditions are such that the *setacl* call will succeed. (See Figure 2 and relevant assertions in the text.) Thus, when testing the -s option of *setacl* the input generated by the valid class of input is as follows:

I1: *setacl -s* where the process owner is the object owner.

Note that similar input will be generated for other applicable options to *setacl*.

Another valid condition for *setacl* is the one in which the calling process is privileged (TE0203).

I2: *setacl -s* where the process is not the owner of the object
and the process has the P_OWNER privilege.

The corresponding output for the above input would be the indicators of successful completion.

The invalid class includes the condition where the process owner is not the owner of the object and the process does not hold the P_OWNER privilege (TE0204). This condition also mandates a specific class of output (TE0204).

To continue with the *setacl -s* case:

I3: *setacl -s* where the process is not the owner of the object
and the process does not hold the P_OWNER privilege.

O3: return code -1, *errno* = NOPERMIT

Intuitively, one would expect that for every valid class of inputs and outputs, there would be at least one corresponding member of an invalid input and output class. Such will be the case except for functions which have no possible invalid inputs. Thus, the tests for completeness continue during the test case specification process.

Similarly, boundary analysis is performed for inputs which consist of scalar values.

5. Problems and solutions

5.1 Document stability and revisions

The degree to which the mapping method is tied to the design documentation is a two-edged sword. The advantages are clear. However, it also means that every change in documentation necessitates corresponding changes in the mapping structure. Where the mapping is produced manually, and the connections are on paper, this can be very time-consuming. It is also necessary to track documentation changes carefully. The inclusion of "change bars" in revisions of documents is a bare minimum expedient.

5.2 Rigorous process is also tedious

The process of exhaustively dividing documents into assertions is time-consuming. Furthermore, the mapping of assertions requires that the tester scan assertion lists for matching assertions, which can require time and patience.

The positive side is that the documents are, by the very nature of how they are used, subjected to the most exhaustive review process possible. A large number of design errors were caught by this process, and prevented from propagating into the code.

6. Results

The foregoing method provides the following benefits:

The approach:

1. validates requirements and design document consistency and integrity, providing an argument for the correctness of the implementation
2. establishes traceable relationships from the TCSEC through to the test elements
3. identifies requirements and design errors early in the product development cycle as missing or mismatched links between assertions appear
4. leads to test suites which are complete in their coverage when the mapping approach is supplemented with "classical" test techniques
5. provides for integrity of change control via the traceable relationships between the base documents and the test suites themselves

The assertion-mapping method has been demonstrably effective. On one feature, the analysis of 167 requirements assertions led to the discovery of 7 contradictions and 27 omissions in the design phase. These discrepancies were those that remained even after completion of a formal document review cycle. It is clear then, the method is far more effective at detecting errors than reviews and walkthroughs alone.

Errors which survive into the final object code are generally thought to be orders of magnitude more costly than those which are resolved during the design and requirements phase. Therefore we are convinced that such a structured approach to test design is worthwhile.

7. *Acknowledgements*

Credit for the development of the above method extends well well beyond the named authors. In particular, Theresa Coulson, Joanna Shih, and Brian Weis were involved with these ideas from their inception. The entire test team also contributed greatly to the refinement of the method.

8. *References*

1. Department of Defense. Trusted Computer System Evaluation Criteria, DOD 5200.28-STD, December, 1985.
2. The Institute of Electrical and Electronics Engineers, *IEEE Standard for Software Test Documentation ANSI/IEEE 829-1983*, The Institute of Electrical and Electronics Engineers, Inc., New York, 1983.
3. Glenford J. Myers, *The Art of Software Testing*, John Wiley and Sons, New York, 1979.

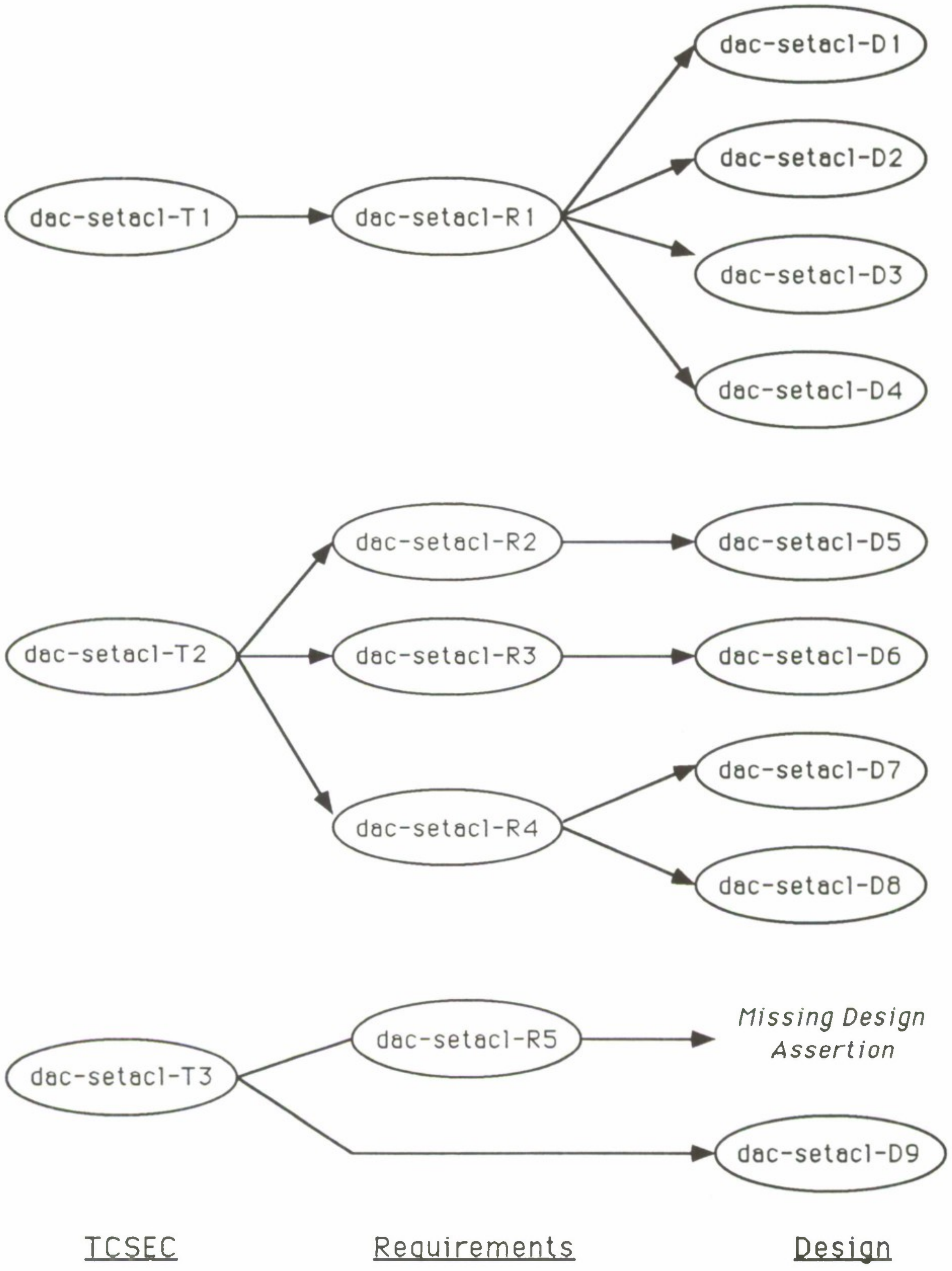


Figure 1: Example of Mapping Rules Application

Figure 2.

TEST ELEMENT MAPPING			
TCSEC	Requirements	Design	Test Element
dac-setacl-T1	dac-setacl-R1	dac-setacl-D1	TE0201
		dac-setacl-D2	
		dac-setacl-D3	
		dac-setacl-D4	
dac-setacl-T2	dac-setacl-R2	dac-setacl-D5	TE0202
	dac-setacl-R3	dac-setacl-D6	TE0203
	dac-setacl-R4	dac-setacl-D7	TE0204
		dac-setacl-D8	
dac-setacl-T3	dac-setacl-R5		None (mapping omission)
		dac-setacl-D9	TE0206

SECURITY TESTING: THE ALBATROSS OF SECURE SYSTEM INTEGRATION?

Susan H. Walter
Grumman Data Systems, Washington Operations
6862 Elm Street
McLean, Virginia 22101
Walter @ Dockmaster.ARPA

ABSTRACT

Security Testing is one of the most important phases of secure system integration because it verifies security functionality and exposes unforseen vulnerabilities. Once security functionality is confirmed, any identified vulnerabilities can be minimized through countermeasures that in turn provide a more secure system. Security testing is discussed in four phases: Planning, Preparing, Executing, and Reporting. Each phase is discussed in detail.

INTRODUCTION

The Department of Defense Trusted Computer System Evaluation Criteria (TCSEC) [1] requires that "the security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation." Taken in the strictest sense, the TCSEC only applies to systems that are undergoing evaluation by the National Computer Security Center (NCSC). Still, some system integrators are embracing this technology to provide the acquisition agency with some assurance that the security mechanisms of the system perform as designed. This paper focuses on the systems integration problem rather than the NCSC evaluation process.

Security Testing is one of the most important phases of secure system integration since it verifies security functionality and exposes unforseen vulnerabilities. Once security functionality is confirmed, any identified vulnerabilities can be minimized through countermeasures that in turn provide a more secure system. Therefore, security testing must be thorough to be successful. By thorough, I mean that it encompasses all the Trusted Computing Base (TCB), it verifies that all security requirements are met, and it exposes system vulnerabilities. The term Trusted Computing Base is used here to mean that portion of the system responsible for enforcing the security policy.

Unfortunately, security testing is time consuming and occurs at the end of the integration process when schedules are tight and tempers are on edge. Just as the ancient mariner [2] saw shooting the albatross as the solution to his problems, the system integrator often sees cutting down or eliminating part of the security testing process as the solution to schedule problems. But as the ancient mariner found out, this will only make things get worse because without adequate security testing, there is no assurance that the system will protect data appropriately. Often, the fallacy of cutting short security testing only becomes apparent after a serious breach of security that threatens the system's existence, the owner's integrity, and the developer's future business prospects.

There are four phases of security testing: Planning, Preparing, Test Executing, and Reporting. Our experience has shown that actual test execution takes up only 10 percent of the time. The planning, preparing and reporting phases make up the other 90 percent of the time.

Throughout the paper, the terms contractor and customer are used. Contractor refers to the system integration entity conducting the tests and customer refers to the entity requesting the tests.

PLANNING

The Test Plan is the single most important document associated with testing. It provides all the information necessary about how the tests will be executed and serves as the testing agreement between the contractor, the customer, and the system and/or accreditor. The Test Plan contains the data derived from the five major tasks of the planning phase: Defining the Process, Defining the System, Defining the Test System, Defining the Test Team and Schedule and finally, Receiving Approval to Proceed.

The first task during security test planning is to Define the Process. Defining the process means answering the question "What is the purpose of this security testing?". Specifically, this task determines whether the system is being tested as a part of a certification, an accreditation, or both.

Certification is a technical evaluation of a system's security features and other safeguards, made in support of accreditation, to establish the extent to which a particular computer's design and implementation meet a set of specified security requirements [1]. Typical certifications done in support of accreditation are administrative, procedural, physical, emanations, personnel, communications, and computer-based.

Certification should not be confused with the evaluation process that NCSC performs on Commercial-Off-The-Shelf (COTS) products. The result of the NCSC evaluation process is a product that is rated by the NCSC to provide security features at a certain level (C2, B1, B2, etc.), each of which is defined by the Orange Book. The use of an evaluated COTS product is a bonus in the computer-based certification, but will not guarantee that the implementation of that product should be certified in support of accreditation. The computer-based certification involves all the hardware, firmware and software, not just the evaluated product.

Accreditation is a managerial decision that the system is "safe" enough to process sensitive information in a specific operational environment, based on a comprehensive evaluation of the system's security design including hardware, firmware, software, configuration, and implementation. Accreditation also considers procedural, administrative, physical, TEMPEST, personnel, and communications security controls [1]. Accreditation is granted by a Designated Approving Authority (DAA). The official serving as the DAA depends on the sensitivity of the data present on the system.

This paper is focused on the computer-based certification concentrating on the security features of the hardware, firmware and software. The specified security requirements for this certification could be the TCSEC [1], the customer's or DAA's Security Policy, or any other design requirements document for the acquisition.

Defining the Process also includes defining the test director's and tester's jobs, determining the administrative details of testing (such as how test discrepancies will be handled), and defining testing terms (such as stress and throughput) that may be included in the Data Item Description (DID) used for the Test Plan. Two types of test discrepancies must be distinguished in the Test Plan. Type 1 discrepancies occur when the test procedure is wrong and the system is functioning correctly. Type 2 discrepancies occur when the test procedure is correct and the system is functioning incorrectly. These two types are handled differently during testing. For Type 1, the test procedures simply need to be updated. For Type 2, there is a system problem that could result in a test failure. Because of the significant difference between these two events, it is best to separate them in some way. We did this by calling Type 1 events deviations and Type 2 events discrepancies. Test deviations need only be noted so that the test procedures can be updated. Test discrepancies, on the other hand, must be investigated further. It must be determined in the test plan how each of these events will be handled during testing.

The second task during the planning phase is to Define the System <u>in security terms</u>. This will decide the focus of the tests in terms of test objectives. At first glance, it may seem that the system has already been defined. Isn't that what is done during system design? The answer to that question is a qualified "yes" [3]. Without a security model or security architecture document, the system has not been defined IN SECURITY TERMS. Large mainframe integration projects typically involve large numbers of COTS products in addition to the operating system(s) and security package(s). Except for specially designed security packages, chances are these packages were selected based on their functional or performance capabilities with little or no regard for any security or vulnerability embedded within them. Each of these software packages must be analyzed to determine how they interface with the operating system and/or the security package, and whether they include any security or introduce any vulnerability on their own. This process must be followed even if the security package is an NCSC-evaluated product. All COTS packages that include security must be considered part of the Trusted Computing Base (TCB) and must be tested during the security testing process. During the testing process, it must be shown that vulnerabilities are constrained by the TCB. Only when this information has been accumulated, can the TCB be identified. THE DEFINITION OF THE TCB IS CRUCIAL TO THE SUCCESS OF TESTING. Without proper definition of the TCB, there is no assurance that security testing will cover all the necessary areas.

Defining the System also includes determining the system boundary. The system boundary is the line between what is a part of the system and what belongs to the rest of the world. System boundary determination hinges on specifying the interface between the system and the outside world. Everything on one side of the interface is within the system boundary; everything on the other side of the interface is not. This interface is enforced by external security controls, and as long as those controls are in place, testing will determine if the internal controls protect the system information against the specified threats. If something bypasses the external controls and enters the system without authorization, or if outside forces threaten the system in an unanticipated way, then all bets are off [4]. System boundary definition is important to ensure that during security testing the system will not be expected to protect more than it is really responsible for.

The third task of the planning phase is to Define the Test System. This step includes defining the exact suite of hardware and software, how the security parameters of the COTS software will be set, all test tools, all test-unique modifications to the system, and any specialized personnel who will be used during testing. A justification for any differences between the test system and the operational system as previously defined should be included. If the test system does not resemble the operational system closely enough, the security tests may be worthless in terms of actual functionality and vulnerability definition.

In addition, it should be stated clearly that the test team needs a dedicated system for the entire time allotted for testing. Users on the system while tests are being run may invalidate the results. In addition, the system cannot be used for any other purpose during the testing cycle because this could invalidate the tests as well. The configuration of the system (hardware and software) must remain constant and under complete control of the test team.

The fourth task during the planning phase is to Define the Test Team including the Test Director, the testers, and any observers, such as customer, certification or accreditation representatives and to define the Schedule. Continuity will be provided between all phases of testing by having these people designated in advance so they can participate in the writing and reviewing of the Test Plan. In addition, this ensures that all personnel have or can get the proper security clearance and training that will be needed for on-site testing and are available for travel should this be necessary.

Included in the fourth task is defining the schedule. Within this schedule time must be allocated for setting up the system, running the tests, analyzing and verifying the results, and retesting. These steps must be done while the test team has the system. Ample time should be allowed for false starts as well. Nothing ever runs smoothly during testing, but planning and coordination go a long way to alleviating this problem.

The final task of the planning phase is to receive approval to proceed. As previously stated, the Test Plan serves as the agreement between the contractor, the customer, and the system certifier and/or accreditor about how testing will be done and what will be involved. After approval, the testing process can proceed with an agreed upon plan to follow.

PREPARING

The second phase of security testing is Preparing for testing. This involves training the testers and writing and dry running the procedures.

Security test personnel are not likely, nor desired to be, the system's programmers or implementers. The team will require training and hands-on experience with the system to provide them with the knowledge necessary to produce detailed test procedures. The optimum situation would be to have the security testers involved during the actual implementation of the system. Participation in this process would expose them to the expert knowledge of the system programmers on the integration of the system, and could avoid costly mistakes during testing caused by not understanding the entire scope of the system.

Writing the test procedures requires several up front decisions. First, should the tests be independent or dependent? Each of these methods has advantages and disadvantages. Independent test procedures make for easier testing. If something fails within one test, that test can be rerun with minimal effort, and test replication is critical. However, dependent test procedures do not require as much set up because the tests can build upon one another to get the system into the state needed for a specific test. Second, does the order of execution of tests matter? This will depend on whether the tests are independent or dependent. If the tests are dependent, then the order of execution is extremely important. If they are independent, the order may not matter from a strictly functional viewpoint, but there may be some aesthetic value to starting with the simple things and working up to the more difficult ones. From an observer's point of view, it would be easier to follow the tests if they started with identification/authentication rather than something buried within the TCB such as auditing.

The test procedures themselves will be easier to understand and execute if they are presented as an organized package. It is helpful to have the test objective for each particular test stated at the beginning of the procedure. Following this, a setup section containing any special system requirements for this test is necessary. The detailed procedures follow. Detailed procedures are needed because the tests must be repeatable and there must be a defined path for each test to allow this [3]. The detailed procedures should include expected results for each test to serve as pass/fail criteria so that it can be determined immediately if a deviation or discrepancy has occured. The last item included in the procedure should be the test termination items. These include collecting the audit trails from the system, a list of expected audit events for verifying the audit trail, verification steps, ensuring all logs associated with the test are complete, and restoring the system to its pretest state, if necessary.

Once the test procedures have been written, they should be dry run on the test suite. The dry run gives the testers further system exposure, ensures that the procedures are written correctly, and irons out the test execution. The importance of dry running the test procedures is often overlooked when schedules get tight and program managers are searching for ways to cut the testing time. If the procedures are not dry run, there is no assurance that the tests will work correctly during actual testing. Valuable test time could be lost determining whether the test procedure was wrong, or the system was functioning incorrectly. Incorrect test procedures used during actual test execution generate unnecessary paperwork because all test deviations and discrepancies must be documented and explained in the test report. This wastes time and does not present a good view to the customer, the certifier or the accreditor.

EXECUTING

The test execution phase is the culmination of the previous efforts, and will typically take much less time than the previous phases (but usually more time than is allotted!). The test execution phase begins with a pretest meeting between the contractor, the on-site personnel, the customer, the certifier, and the accreditor (if present). All personnel involved with the testing process should attend this meeting since all areas of testing will be discussed. This forum will give the customer and on-site personnel an idea of what will occur. Items to be discussed include points of contact, administrative details such as badges, system configuration, and necessary special equipment for testing.

Before testing can begin, the exact system configuration including hardware and software should be documented including not only the type of hardware and software but also the setting of any variable software parameters. If any changes are made to the configuration by the testers during testing, these will also be documented. Configuration changes must be considered carefully to ensure that they do not invalidate any completed tests.

Once testing begins, test logs should be kept for each test in a format containing the test number and title, the names of the testers and observers, the date, and a check list for the test steps. Anything unusual that happens during test execution can be written on the log sheet so it is not forgotten or overlooked if things get hectic. These logs are invaluable in writing the Test Report and provide detailed information about the actual test execution that could be otherwise lost.

During testing, all test deviations and discrepancies should be documented on discrepancy forms in the agreed upon format contained in the Test Plan. These events need to be documented when they occur so that details are not omitted. Every piece of information associated with the event is necessary. This is imperative for repeating the test step or entire test, if necessary. It is very difficult to decide if a detected problem has been fixed when there is not enough data to attempt a recreation. Test deviations also should be noted to ensure the test procedures can be corrected since often, the test procedures will be used more than once.

At the completion of every test, the test data (audit trails, test logs, etc.) should be analyzed. This analysis will take a significant amount of the allotted test time and must be included in the test schedule. The audit trails must be verified against the expected results to find if the system audited all appropriate events including authorized accesses, unauthorized accesses, invalid logons, etc. Analysis of the audit trail also will show if anything unexpected happened that was not apparent to the testers. Only after this analysis, can it be determined if the system passed or failed a test.

Occasionally some testing event or the analysis of the audit trail will be ambiguous. If this happens, a retest may be necessary. It is important to decide if this is necessary before another test begins since data may be needed from the system that would be invalidated by the starting of another test. Additionally, if a retest is necessary, the system will already be in the correct setup state for that test if another test has not begun.

During testing, daily status reports are important outputs and, with team coordination meetings, serve two functions. First, these reports provide the customer with a feel for how things are going, and second, provide good documentation to support the writing of the test report. These daily reports summarize everything that occurs relative to testing during the day. In addition, they include all tests initiated or completed, all discrepancies and deviations noted, and a status of pending discrepancies. A copy of all discrepancy forms should be included with the status report at the end of the day.

REPORTING

The Test Report provides the customer with the results from the testing and all supporting documentation. There are five sections to a complete Test Report. The first section gives the system configuration at the beginning of testing and any changes that were made during testing. Test Logs can be attached to support these assessments.

The second section of the Test Report contains a summary of each test and its outcome. This includes when the test was run, who ran the test, who witnessed the test, all discrepancies and how they were resolved, all test deviations, and a general assessment of whether the system passed or failed the test.

The third section of the Test Report contains all the supporting documentation from each test including the test logs, all discrepancy forms, and the audit trails produced from the execution of the tests.

The fourth section of the Test Report contains a copy of the daily status reports generated during testing.

Finally, the last section of the Test Report contains the conclusions and recommendations based on the test execution outcome. This section should be considered the managerial overview of the results and is the most important part of the Test Report. It will provide management with the data they need to decide whether the system should be certified and/or accredited. In addition, recommendations can be made for countermeasures to mitigate the vulnerabilities discovered through the testing process. For customer convenience, this section can be published separately from the rest of the document, since the logs and audit trails can be extensive.

CONCLUSION

Security testing verifies security functionality and exposes vulnerabilities to provide a more secure system. It involves four phases: planning, preparing, executing, and reporting. All of these phases are equally important to the success of the testing, and therefore, none of them can be sacrificed to make up schedule losses.

REFERENCES

1. National Computer Security Center, Department of Defense Trusted Computer System Evaluation Criteria, DoD 5200.28-STD, Maryland, 1985, pp. 13, 111-112

2. Samuel Taylor Coleridge, "The Rime of the Ancient Mariner", in Best Loved Poems To Read Again and Again, New York: Galahad Books, 1988, pp. 81-111

3. C.J. Haley and F.L. Mayer, "Issues on the Development of Security Related Functional Tests", in Proceedings of the 8th National Computer Security Conference, 1985, pp. 82-85

4. M. Gasser, Building a Secure Computer System, New York: Van Nostrand Reinhold Company, Inc., 1988, Ch. 3, pp. 19-21

LOW COST OUTBOARD CRYPTOGRAPHIC SUPPORT FOR SILS AND SP4

B.J. Herbison
Secure Systems Architecture
Digital Equipment Corporation
85 Swanson Road BXB 1-2/D04
Boxborough, MA 01719-1326
Herbison@Ultra.ENET.DEC.COM

Abstract

This paper describes a method for developing relatively inexpensive cryptographic hardware that could be used, along with software, to efficiently provide LAN and WAN confidentiality and integrity services. The paper describes the design of the hardware and how it could be used by software to support different communication security architectures.

Introduction

The need for secure communication between computer systems is increasing as the number of network and distributed applications increases. The Digital Distributed System Security Architecture[1] assumes that secret key cryptography is 'inexpensive and pervasive'. However, there are several tough constraints on the development of communication security systems, even for commercial environments. Some common constraints are the existence of multiple communication security standards, the need to provide communication security for previously manufactured computer systems, and a desire for low cost solutions.

When cryptographic support is being designed for one system, the least costly solution us to integrate the cryptographic support hardware into the system. However, there are a variety of reasons why this isn't always the best solution:

- The cost of integrating the cryptographic hardware increases the cost of designing each new system. If a large number of different systems need to be protected then the development costs increase.
- An integrated solution can't be used to protect previously manufactured equipment or equipment manufactured by other companies.
- Export and import regulations restrict the potential markets for a system with integrated cryptographic support.

This paper proposes a scheme that could be used to build an inexpensive cryptographic device that can be attached to a variety of systems to provide LAN and WAN communication security. Only software changes would be required to support this device. The device would be attached between a system and its LAN connection.

Communication Security Architectures

Communication Security standards are being developed for the Data Link, Network, and Transport layers. The IEEE 802.10 Working Group is developing a *Standard for Interoperable LAN Security* (SILS)[2, 3, 4] that provides security at the Data Link Layer. The Secure Data Network Systems (SDNS)[5] Protocol and Signaling Working Group has developed *Security Protocol 3* (SP3)[6] and *Security Protocol 4* SP4[7] to provide security at the Network and Transport layers respectively, and SP4 has been proposed as an ISO standard[8].

In addition to these standards, a system may also want to support proprietary or application-specific security protocols.

All references to SILS in this document are based on the 2 June 1990 draft of the SILS Secure Data Exchange protocol. Changes to the SILS draft before it becomes a standard could prevent the device described in this document from being developed. In any case, it is not possible to complete the design of any device that supports SILS until SILS becomes a standard.

Design Goals

No one communication security product will work in all environments or satisfy all requirements. The following list contains the requirements that led to the design described in this paper:

- The design must support both SILS and SP4 (both SP4-E and SP4-C). Providing LAN security implies that the device does not need to support systems without a LAN connection.

  Since SP3-N is the same as SP4-E, support for SP4 implies support for SP3-N.

- The communication security hardware must be as inexpensive as possible.

- Secured communication must not be significantly slower than unsecured communication. The effort of providing security must not cause the system to be overloaded or cause messages to be delayed for significant period of time.

- In order to avoid developing a large number of hardware devices, one device must be able to support a wide variety of computer systems, including systems designed before the communication security device was conceived.

Note that these goals do not require that communication security be provided in a manner that is transparent to the system. Unlike Digital's Ethernet Enhanced-Security System[9], the design in this paper requires software on the system to assist in the process of providing communication security. This implies that the system is trusted and the device provides no protection against covert channels.

The following are desirable requirements for inexpensive hardware:

- Perform only simple operations and minimize the number of states.

- Use as little memory as possible for storing messages and other information.

- Make the user interface as simple as possible with few controls or indicators. If possible, allow no user interface.

Our Scheme

Our solution is to build a simple outboard cryptographic device that handles only encryption and decryption. Anything that cannot be handled easily is left to software on the system. The software on the system handles key management, consistency checking, management, and all the special cases and other details of the communication security protocols. The device has no user interface—it is totally controlled through frames sent by the software on the attached system.

The device encrypts, decrypts, and calculates integrity check functions on frames as they are transmitted by the system, and on frames from the network as they are received by the system. All operations occur at the rate of data transmission on the LAN—a delay may occur but there is no loss of throughput. In addition to providing high performance, this also reduces the frame buffer requirements.

The device recognizes several frame formats (both SILS and SP4) in frames passing through the device and encrypts or decrypts parts of the frame. If a frame doesn't contain a recognized format, then the frame is passed through unmodified. If the software discovers that a decrypted frame shouldn't have been decrypted, the software reverses the operation by using 'loopback' mode.

Encrypted Keys

One important feature of our design is the method for selecting the key that needs to be used to encrypt or decrypt a frame.

SILS and SP4 both provide mechanisms for identifying the key for an association. The SP4 message format currently contains a variable length key identifier field, and the current SILS Secure Data Exchange (SDE) message format contains a four byte Security Association Identifier (SAID) field. In addition, SILS SDE also specifies an optionally supported, variable length Management-Defined field to allow "the transfer of information that may facilitate the processing of the PDU"[4].

When key management sets up an association between a pair of systems, both systems are given the keying information for the association. Also during association establishment, the systems select and exchange values for the fields described above (key identifier or SAID and Management-Defined, depending on the protocol in use). Each implementation decides how it wants to interpret these fields, and the ability of systems to interoperate is not affected at all by the manner in which these fields are used.

One common implementation approach is to include in the fields an index into a table containing encryption keys. A drawback of this approach is that it forces the cryptographic device to store a potentially large table of keys. To avoid this storage, our scheme essentially stores the association key in this field.

Each system using our scheme has a key-encryption key (KEK) that the system generates randomly when it boots. The system loads this KEK into the cryptographic device, but doesn't distribute the KEK to any other system.

When the key management software in the system sets up a pairwise association with a remote system, it encrypts the association key under the KEK and sends it to the remote system as part of the key identifier—the remote system places the key identifier in the SP4 key identifier field or the SILS Management-Defined field of frames sent back to the system that generated the key identifier. The distribution of the encrypted association key (the key identifier) is totally separate from the determination and exchange of the association key between the system—the key exchange operation happens in the manner described in the appropriate key management document.

When the device receives an incoming frame containing an encrypted key in the SP4 key identifier field or the SILS Management-Defined field, it decrypts the key with the KEK and uses the result to decrypt the frame.

Along with the encrypted key, the key identifier also includes a control field that identifies how the key is used. For example, the control field can be used to select between a mode that provides both confidentiality and integrity and a mode that provides cryptographic integrity but not confidentiality.

The security of a systems communication depends on the secrecy of the system's KEK, and security will be compromised if the KEK is disclosed or cracked. The system never releases the KEK, but the key encryption algorithm has to be strong enough to resist a known plaintext attack by the systems with which it communicates. (This same caveat also applies to the method used to distribute keys in any cryptographic system that automatically distributes association keys.)

This mechanism of placing encrypted keys in the frames works for all SP4 frames and all SILS where the remote system supports the Management-Defined field. The device can be used, but not as efficiently, in situations where the remote system does not support the SILS Management-Defined field.

Transmit Processing

The transmit processing performed by the device is a simple generic scheme.

When the system wants an outgoing frame to be encrypted, the system prepends some special information to the frame and then transmits the frame. The special information consists of a *trigger* value that indicates to the device that the frame requires encryption, *control* information that describes how to encrypt, the *key* to be used for encryption (encrypted with the KEK previously loaded into the device) and a *count* of bytes to skip before the encryption starts.

The trigger value is a value that never occurs at the start of any other type of frame transmitted by the system. The choice of a trigger is a local issue that doesn't affect interoperability, but one way to guarantee uniqueness is to reserve an IEEE 802.2 protocol identifier value.

The device processes frames by looking for the *trigger*, saving the *control* and *key* information, and then transmitting *count* bytes of the frame unmodified followed by the rest of the frame encrypted with *key*. If the Data Link protocol requires a frame check sequence (FCS), the device must remove the FCS from the frame received from the system and add a correct FCS to the transmitted frame. The transmit processing is shown in Figure 1. The system needs to build the frame so that the transmitted portion contains a complete Data Link frame.

In order to reduce the need for buffering, the cryptographic hardware encrypts the frame at the transmission speed of the frame. In addition to simplifying the device and reducing the memory required for buffer frames, this can also reduce the latency of the device by allowing the device to start transmitting the encrypted frame before the frame has been totally received by the device.

As an example of how this processing works, if the system want to transmit an encrypted SP4 TPDU in an Ethernet frame, the system first builds a SP4 SE TPDU, omitting the encryption step. The Network and Data Link headers are added to build a complete Data Link frame. The special information described above is added to the frame, with the *count* value equal to the total length of the data link headers, the Network IP headers, and the SP4 clear headers.

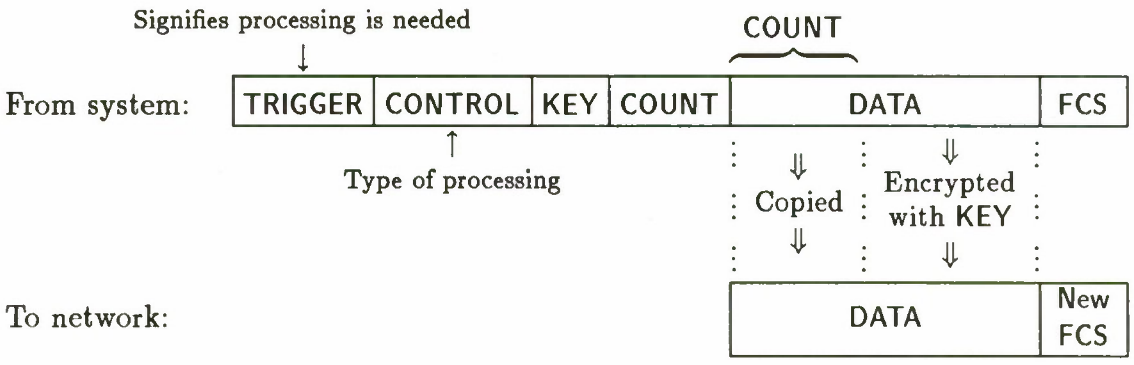


Figure 1: Generic Transmit Processing

When the result is transmitted, the device removes the special information and encrypts the appropriate portion of the frame. This operation is shown in Figure 2.

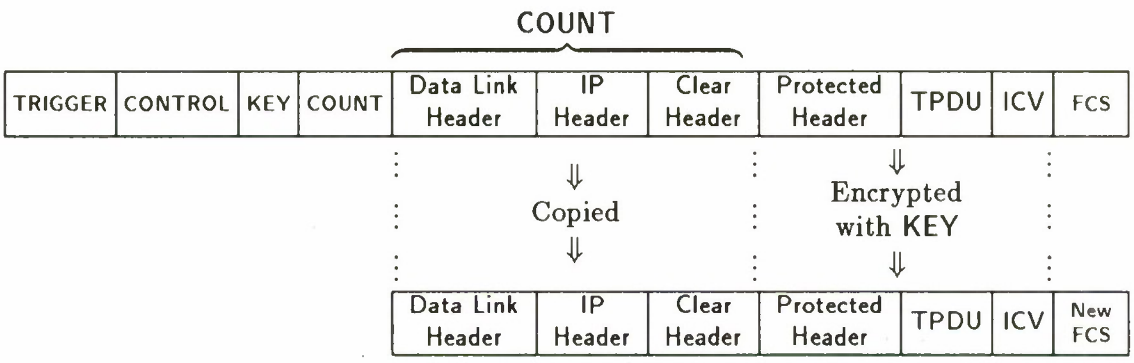


Figure 2: SP4 Transmit Processing

In addition to just encrypting the frame, the device can also calculate an integrity check value (ICV) for the frame and include the ICV in the frame before encryption. If confidentiality is not provided, the encrypted key would be used calculate a cryptographic integrity check function. It would be possible to include two encrypted keys in the frame, one for confidentiality and another for a cryptographic integrity function.

Note that this is a generic processing scheme that isn't limited by any particular communication security protocol. The encrypt and forward process works efficiently even for communication security protocols that cannot be handled by the device for received frames.

Any frame that does not start with the reserved *trigger* value is passed through the device to the network without any modification.

Receive Processing

The processing of received frames is not as generic as the transmit processing. Decryption is only performed if the frame contains a format that is recognized by the hardware.

Frames received from the network are decrypted by the cryptographic device if all of the following conditions are met:

- The frame contains data encrypted using a recognized and supported communication security protocol (i.e., SILS or SP4).

  For SP4, this involves checking the Data Link, Network, and Transport layer headers in the frame. For SILS, only the Data Link header needs to be checked.

- The frame has not been segmented by the network layer during transmission. This only applies to frame containing SP4 TPDUs.

- Once located, the key identifier contains a valid control field.

If one or more of these conditions is not met, the frame is passed on to the system without decryption and the frame is handled by software on the system. If all of the conditions are met, the device locates the encrypted association key, decrypts the key with the KEK, and decrypts the rest of the frame with the association key. As with transmit processing, the device must regenerate the FCS field for any frame it modifies.

In order to process the frame at the rate of Data Link transmission, the device needs to decrypt the encrypted association key and prepare to use the association key before the first block of the encrypted data needs to be decrypted. The device can start decrypting the frame and transmitting it to the system before the entire frame is received from the network, reducing both the buffer requirements of the device and the delay introduced by the device.

This processing is designed to successfully process most of the incoming frames. It handles the most expensive part of the processing (decryption) for frames that can easily be detected and processed by the hardware.

Unlike the encryption operation, no fields are stripped from an incoming frame when the frame is decrypted. The fields must be left intact because they contain information essential for the processing of the frame by the system. In order to determine how to process a frame, the system needs to know exactly how the device processed the frame.

For example, the device would recognize a SILS frame (as defined by the current SILS draft) based on the contents of the SDE Designator field. The CONTROL field is then checked to determine if it contains a valid value. If it is valid, then the encryption key in the frame is decrypted and used to decrypt the rest of the frame. This process is shown in Figure 3. The CONTROL and KEY fields together comprise the SILS Management-Defined field.

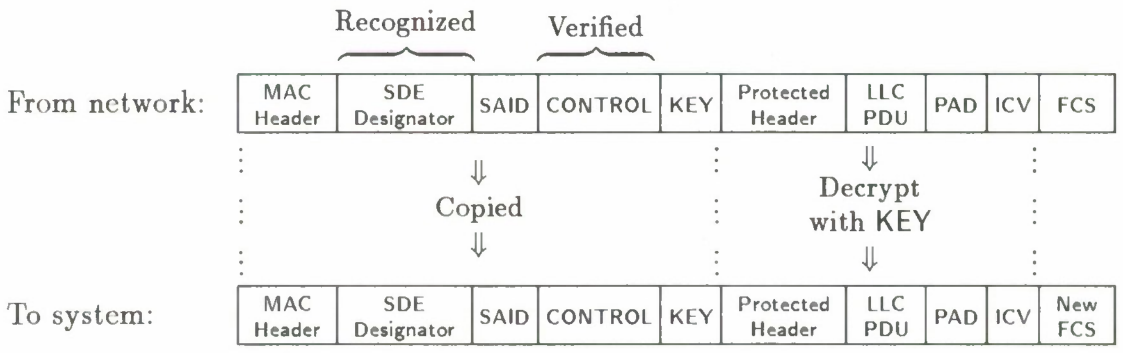


Figure 3: SILS Receive Processing

Address Recognition

Note that in the preceding description of receive processing, no mention was made of checking the addresses of received frames. If the frame contains a SILS or SP4 message that contains an appropriate control field, the device will decrypt the message no matter what address information occurs in the frame. If the frame is addressed to a different system, the decryption will be done with the wrong encryption key.

However, this incorrect decryption is not a problem in most situations. Since the frame is not addressed to the system, the system will ignore the frame and so it doesn't matter that the frame was processed by the device. There is no performance penalty because the device operates at the same speed as the LAN, and the system sees no more frames than it would if the device was not present. The device does not need to do any address filtering and may therefore be simpler. (There are advantages to filtering. Filtering does reduce the number of frames received by the system, which can improve the performance of some systems.)

Routers are one situation where improper decryption can cause problems. Routers receive and transmit frames with a Data Link address of the router and with a Network layer address of the intended recipient. Since SP4 provides end-to-end security, a router does not have the correct key to decrypt the SP4 TPDUs passing through the router. Any attempt to decrypt the frame would be incorrect and the decrypted frame would have to be re-encrypted before being sent to the next system. Since the cryptographic device is designed to detect and decrypt SP4 TPDUs, a potential problem exists.

A solution to this problem is to allow the processing of incoming SP4 frames to be disabled. Routers will disable SP4 processing to prevent incorrect decryption of frames passing through the router. This solution causes the device to work correctly for routed frames, but requires the system to perform extra work when SP4 is used to communicate directly with the router. If the router is not used as a general purpose system, but rather only acts as a router, there should be little communication directly with the router. (If the router doesn't use SILS, the simplest solution is to not connect a cryptographic device to the router.)

Loopback Mode

In addition to transmit and receive processing, the device also supports loopback processing. In loopback processing, a frame is transmitted by the system, received and encrypted or decrypted by the device, and returned to the system.

Loopback mode has a variety of uses:

- Loopback mode can assist the system in performing encryption and decryption operations that the device can't handle in the course of normal processing. These operations include:

  Network layer Segmentation: If a SP4 TPDU is segmented into multiple parts by the network layer during transmission, the TPDU will arrive at the system in multiple Data Link frames. In this case the device will not be able to correctly process the segments (since the key will only be included with the first segment). The device recognizes the segmentation and does not attempt to modify any frame that contains only a segment of a TPDU.

  When this occurs, the system software reassembles the segments and uses loopback mode to decrypt the complete TPDU.

Double Encryption: The device is designed to perform only one encryption or decryption operation for each frame. If a system is configured so that both SILS and SP4 protection are applied to the same data, the device will not be able to perform both operations at once.

If this occurs, the system must use loopback mode to perform the SP4 encryption or decryption, and the device will perform the SILS encryption or decryption when the frame is transmitted.

SILS without Management-Defined Field: When communicating with a system that doesn't support the use of the Management-Defined Field, there isn't enough room in the SILS clear header to include the encrypted key.

In this situation, frames received from the remote system must be decrypted using loopback mode.

Other Communication Security Protocols: When communication security protocol other than SILS or SP4 are used, received encrypted frames are not processed by the device but must be decrypted using loopback mode.

In each of these cases, performance is lower than in cases where the device can perform all processing as frames are transmitted or received. However, the use of loopback makes the processing possible without requiring the system to perform the encryption in software.

- Loopback mode can be used to decrypt SP4 frames addressed to routers. The need for this is discussed in the previous section.
- Loopback mode can be used to provide encryption and decryption services for application-specific security measures. An example of an application that provides it's own security is encrypted mail as defined by Internet RFC 1113[10, 11].
- Loopback mode can be used to provide special cryptographic support for the key management software on the system. In particular, a loopback operation should be provided that encrypts association keys under the KEK of the device.
- Loopback mode can be used for any other cryptographic operations that system wants to perform (e.g., file encryption).

To simplify the device hardware, the transmit and loopback formats are almost identical and the receive, transmit, and loopback formats share the same control field format.

Additional Comments

Other Communication Security Protocols

The device described in this paper could be modified to support protocols other than SILS and SP4. In particular, it would be useful for to define a mechanism similar to SP4 that could be used to provide security for TCP and UDP frames.

Software Trust

The implementation described in this paper gives the system software total control over all access control decisions and the cryptographic operations performed on transmitted information. The device can't store enough information to enforce any non-trivial access control policy. The software on the system must be trusted to perform the correct operation.

For environments where the software on the system is not trusted, this device will be unsuitable. In these situations a more intelligent cryptographic device, which handles key management and access control decisions, is required. However, even when the cryptographic device is able to make decisions about what associations are allowed, and can force all communication to be cryptographically protected, the system software must still be trusted to send the correct information to each system with which it is allowed to communicate.

Inappropriate Decryption

Depending on the frame formats used by supported communication security protocols, it may not be possible for a simple device to determine precisely which received frames require decryption.

When this occurs, the device implementation can choose to be either conservative or liberal about deciding when to decrypt a frame. If the device is conservative, some frames requiring decryption will not be decrypted as they are received and must be decrypted using loopback.

On the other hand, if the device is liberal, frames that should not be decrypted will be decrypted (but only if the part of the frame that the device interprets as a control field matches a valid control field). When the device incorrectly decrypts a frame, the frame needs to be re-encrypted using the same key. This can only be accomplished if every cryptographic operation applied when a frame is received can be inverted using a loopback operation. It is also important that the processing of each frame be totally determined by the contents of the frame and not any external state.

In either case, the device must operate in a deterministic manner so that the system can determine how the device treated each frame.

In some cases the system may determine that some of the receive processing performed by the device is counterproductive. To handle this situation, the device allows the system to selectively disable the decryption of each frame format recognized by the device.

Hardware Complexity versus Software Complexity

The scheme described in this paper is designed to make the operation of the cryptographic hardware as simple as possible, allowing the hardware to be faster and/or cheaper than more complicated designs. This increases the complexity of the software required to control the device. Furthermore, if the same cryptographic hardware is used to support more than one operating system, the controlling software must be implemented (or at least ported) for each operating system.

The software complexity could be reduced by placing more functions into the cryptographic device. However, when we investigated more complex devices we found them to be significantly more expensive to produce, making it more difficult for communication security to be inexpensive and pervasive. In particular, the device described in this paper can be implemented using small amounts of memory and without using a processor chip. The addition of a processor and firmware would significantly increase the cost of the device, as would adding enough memory to store the state of all associations a system may want active at any one time.

Acknowledgments

I wish to thank Bill Birnbaum, Morrie Gasser, Bill Hawe, and Charles Kaufman for their assistance in developing the ideas presented in this paper and for reviewing drafts of this paper.

References

[1] Morrie Gasser, Andy Goldstein, Charlie Kaufman, and Butler Lampson, "The Digital Distributed System Security Architecture", in *Proceedings of the 12th National Computer Security Conference*, NIST/NCSC, October 1989, pages 305-319.

[2] L. Kirk Barker and Kimberly Kirkpatrick, "The SILS Model for LAN Security", in *Proceedings of the 12th National Computer Security Conference*, NIST/NCSC, October 1989, pages 267-276.

[3] *Standard for Interoperable Local Area Network (LAN) Security (SILS) Part A – The Model*, P802.10A/D1 (Unapproved Draft), 9 December 1989.

[4] *Standard for Interoperable Local Area Network (LAN) Security (SILS) Part B – Secure Data Exchange*, P802.10B/D4 (Unapproved Draft), 2 June 1990.

[5] Gary L. Tater and Edmund G. Kerut, "The Secure Data Network System: An Overview", in *Proceedings of the 10th National Computer Security Conference*, NBS/NCSC, October 1987, pages 150-152.

[6] *Secure Data Network Systems (SDNS) Security Protocol 3 (SP3)*, SDN.301, Revision 1.5, 15 May 1989.

[7] *Secure Data Network Systems (SDNS) Security Protocol 4 (SP4)*, SDN.401, Revision 1.3, 2 May 1989.

[8] *Proposed Working Draft for Addenda to ISO 8073 and ISO 8602 Covering Cryptographic Data Protection*, X3S3/89-50, X3S3.3/89-111R, ISO/IEC JTC 1/SC 6/WG 4, 26 April 1989.

[9] B.J. Herbison, "Security on an Ethernet", in *Proceedings of the 11th National Computer Security Conference*, NBS/NCSC, October 1988, pages 219-225.

[10] John Linn, Internet Activities Board Privacy Task Force, *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encipherment and Authentication Procedures*, Network Working Group Request for Comments (RFC) 1113, August 1989.

[11] John Linn and Stephen T. Kent, "Privacy for DARPA-Internet Mail", in *Proceedings of the 12th National Computer Security Conference*, NIST/NCSC, October 1989, pages 215-229.

LAYER 2 SECURITY SERVICES FOR LOCAL AREA NETWORKS

Richard L. Parker II
The MITRE Corporation
M/S E066, Burlington Road
Bedford, MA 01730
617/271-3076

ABSTRACT

The ISO Security Architecture, ISO 7498-2, was developed using Packet Switched Networks (PSNs) and Wide Area Networks (WANs) as architectural models. Since that time, there have been significant changes in networking practices. Local Area Networks (LANs) have introduced a new range of vulnerabilities that are not present in the Data Link Layer of PSNs and WANs . The point-to-point nature of the Data Link Layer (Layer 2) of PSNs and WANs led to the dismissal of the need for extensive security services at Layer 2. Subnetworks and routing were the focus of the need for inclusion of particular security services at the Network and Transport Layers. However, LANs have introduced subnetworks and routing into the Data Link Layer of many networks. Efforts aimed at providing security services for LANs have found the current Link Layer security service profile in ISO 7498-2 to be deficient. It is necessary to expand this service profile to protect LANs, even in the presence of security services at higher layers in the protocol stack.

INTRODUCTION

In the spring of 1988, preliminary meetings were held to determine interest in security standards for Local Area Networks (LANs). These meetings were initiated by Stanley R. Ames, Jr. and Kimberly E. Kirkpatrick of the MITRE Corporation. More than 40 vendors and users of LANs responded positively. This led to the formation of the IEEE 802.10 LAN Security Working Group, which Kimberly E. Kirkpatrick chairs. This effort is sponsored jointly by the IEEE 802 Technical Committee and the IEEE Technical Committee on Security and Privacy. The working group's charter is the development of Standards for Interoperable LAN Security (SILS).

Since its formation, the LAN Security Working Group has concentrated on development of a Secure Data Exchange (SDE) protocol to be inserted between the Media Access Control (MAC) and the Logical Link Control (LLC) sublayers of the link layer in the ISO OSI Basic Reference Model. The working group has recently begun development of a key management protocol and a security management protocol, as well.

In the course of the development of the SDE protocol, the LAN Security Working Group drew up a list of necessary security services. In large part, this list was based on the attributes of emerging LAN security devices. In this paper, I present an analysis of the attributes of LANs which make these security services necessary. I identify the pertinent attributes and detail the associated security threats. Then, I indicate the security services necessary to counter those threats, giving examples of the benefits of application of those security services, and discussing mechanisms for providing the services.

SECURITY SERVICES UNDER THE ISO SECURITY ARCHITECTURE

ISO 7498-2 identifies five basic security services: access control, authentication, data confidentiality, data integrity, and non-repudiation. These services provide assurance against the security threats of unauthorized resource use, masquerade, unauthorized data disclosure, unauthorized data modification, and repudiation, respectively. This standard also defines the layers within the ISO OSI Basic Reference Model where it is appropriate to apply these services. Appendix B of ISO 7498-2 gives a brief justification for the indicated service placement.

In ISO 7498-2, data confidentiality is the only security service indicated for the Data Link Layer of the ISO OSI Basic Reference Model. Other security services were "not considered useful" at this layer. This paper details arguments for the inclusion of the services of authentication, access control, and data integrity at the Data Link Layer, as well. It is important to note that the arguments presented in this paper are based on changes in networking practices since ISO 7498-2 was completed, not on deficiencies intrinsic to ISO 7498-2 as it was originally conceived. LAN standards have only recently begun to appear in the ISO standards arena (e.g., ISO 8802-2, ISO 88027498-2). Because of changes in of LAN technology, the risks to LANs have become more critical than first considered. High-speed, long distance LANs (e.g., the Fiber Distributed Data Interface, or, FDDI), filtering LAN bridges, and LAN server facilities have increased the range of resources which are vulnerable to abuse. Ring topology networks not only make every Protocol Data Unit (PDU) (e.g., packet, frame) available to every station on the LAN, but require every station on the LAN to receive and then forward every PDU, in order for the LAN to operate properly. These issues have prompted the concerns that lead to this set of arguments. Figure 1 illustrates the differences between the security service profile defined in ISO 7498-2 and the profile proposed for LANs.

	IS 7498/2 Services	IS 7498/2 Services + LAN Services
Layer 7 Application	Authentication, Access Control, Data Confidentiality, Data Integrity, Non-repudiation	Authentication, Access Control, Data Confidentiality, Data Integrity, Non-repudiation
Layer 6 Presentation	Data Confidentiality	Data Confidentiality
Layer 5 Session		
Layer 4 Transport	Authentication, Access Control, Data Confidentiality, Data Integrity	Authentication, Access Control, Data Confidentiality, Data Integrity
Layer 3 Network	Authentication, Access Control, Data Confidentiality, Data Integrity	Authentication, Access Control, Data Confidentiality, Data Integrity
Layer 2 Link	Data Confidentiality	Authentication, Access Control, Data Confidentiality, Data Integrity
Layer 1 Physical	Data Confidentiality	Data Confidentiality

Figure 1

In a specific implementation, a security service can be implemented in any layer at which it is indicated. A service may appear in one layer, more than one layer, or not at all. ISO 7498-2 only indicates where the service can appear, not where the service is required to appear. The security requirements for a particular implementation will determine where the services will be provided. In practice, it is desirable to protect information both at the highest possible point in the protocol stack (i.e., the application layer) and any layers at which subnetworks and routing are implemented.

The ISO Security Architecture was developed using PSNs and WANs as an architectural model. It was assumed that these networks would have a tightly controlled Data Link Layer configuration. In this model, the

HDLC Frame was used to represent the Data Link Layer PDU.[1] It was also assumed that the Data Link Layer of LANs had the same attributes as the Data Link Layer of the model. In fact, while LANs are similar to PSNs and WANs at the Data Link Layer, they also exhibit some of the attributes of the Network Layer of PSNs and WANs. For example, the Data Link Layer of LANs exhibits subnetwork and routing functions very similar to those of the Network Layer. These functions are cited as justification for the Network Layer security service profile, which is the same as the security service profile proposed in this paper for the Link Layer. These similarities and differences are indicated in the following sections as I explore the security-pertinent attributes of LANs.

LAN CHARACTERISTICS THAT NECESSITATE SECURITY SERVICES AT THE DATA LINK LAYER

There are certain characteristics of LANs that necessitate security services at the Data Link Layer. In particular, I will analyze four characteristics of LANs: the manner in which data is transmitted, the manner in which data is received, the nature of LANs' address space, and geographic dispersion of LANs. I will identify the security threats associated with these characteristics. I will then indicate the security services required to address these threats and show how they are applied to LAN data. Finally, I will discuss mechanisms for providing these services.

DATA TRANSMISSION ON A LAN

The manner in which data is transmitted on LANs is one of the attributes that necessitates additional security services at Layer 2. In a LAN's Data Link Layer, data is transmitted on media that is shared by every attached system. Effectively, every PDU is transmitted to every other station on the LAN and the source of a given transmission is difficult to authenticate.

The nature of data transmission at the Data Link Layer on a LAN presents two security threats. First, any station attached to a LAN can transmit to any other station attached to the LAN. There are no implicit controls at Layer 2 on access to a resource attached to a LAN. Second, since it is difficult to identify the source of a given data transmission, one station can claim to be another station. Any station, or set of stations, can be imitated from a single tap into the LAN. The source of a given PDU is difficult to authenticate. These threats to the security of a LAN are known formally as unauthorized resource use and masquerade.

DATA RECEPTION ON A LAN

The manner in which data is received on LANs, is another attribute that necessitates additional security services at Layer 2. Since data transmission at a LAN's Data Link Layer is over commonly accessible media, every PDU is available to all attached stations. A PDU could traverse any station on its way to its destination. This means that while it may be addressed to a specific entity, every PDU is effectively received by every other station attached to the LAN.

The nature of data reception on a LAN presents two security threats, since any PDU could be intercepted by any attached station. First, a station could receive data for which it is not authorized. Second, and worse yet, a station could change the data in a PDU before it is received at its intended destination. On LANs, data for any station, or set of stations, can be received from a single station on the LAN. This is especially significant in LANs employing a ring topology, where every attached system must receive and retransmit every PDU in order for the LAN to function properly. These threats to the security of a LAN are known formally as unauthorized disclosure and data modification.

---

[1] While this simplified model may not represent all possible implementations of PSNs and WANs, it does represent the mapping of many PSNs and WANs onto the ISO OSI Basic Reference model. X.25 Packet Level Interface functions are attributed to the Network Layer. The assumption of tightly controlled configurations, in particular, may seem restrictive, but reflects standard practices in the implementation of secure networks.

LAN ADDRESS SPACE

Assignments within the address space of a LAN are also pertinent to security. Each station interface is permanently assigned a specific address. Since any station interface can be attached to any other station interface through a common medium at Layer 2, LAN addresses must be unique at Layer 2. This means that a station cannot determine, by observation, whether the source address of a PDU is valid or not. There is no hierarchical address assignment in LANs, so any possible link address could be valid on any LAN.

As with data transmission, the nature of address assignment at the Data Link Layer on a LAN presents two security threats. First, any station attached to a LAN can transmit to any other station attached to the LAN. There are no implicit controls at Layer 2 on access to a station attached to a LAN. Second, since it is difficult to identify the source of a given data transmission, one station can claim to be another station. Any station, or set of stations, can be imitated from a single tap into the LAN. The source of a given PDU is difficult to authenticate. These threats to the security of a LAN are known formally as unauthorized resource use and masquerade.

GEOGRAPHIC DISPERSION OF LANS

LANs span vast geographic areas, rendering them vulnerable to eavesdropping or wiretap. This renders them vulnerable to the threats of unauthorized disclosure and data modification. As indicated previously, there is a significant scope of information and access available on a LAN at Layer 2; any station, or set of stations, can be imitated from a single tap into the LAN.

Wiretapping on a LAN presents two security threats. First, a station can receive data for which it is not authorized. Second, and worse yet, a station can change the data in a PDU before it is received at its intended destination. Again, on LANs, data for any station, or set of stations, can be received from a single tap into the LAN. This is especially significant in LANs employing a ring topology, where every attached system must receive every PDU for the LAN to function properly. These threats to the security of a LAN, are known formally as unauthorized disclosure and data modification.

SECURITY SERVICES

In this section, I will describe the type of architecture which requires the indicated security services, describe the security services themselves in detail, and review the formal definition of each service from the ISO Security Architecture. I also examine the application of each service to PDUs at the Data Link Layer on a LAN, making note of the portions of a PDU that are protected by the service.

In figure 2, a LAN has been subdivided into several local segments, or subnetworks, that are interconnected through a backbone network. The subnetworks are effected through the use of bridges, which pass a PDU between a subnetwork and the backbone network only when that PDU is directed from a station on one side of the bridge to a station on the other side of the bridge. Some of the subnetworks have been designated as protected subnetworks, i.e., subnetworks that are safe from attachment of unauthorized stations, as opposed to unprotected networks.

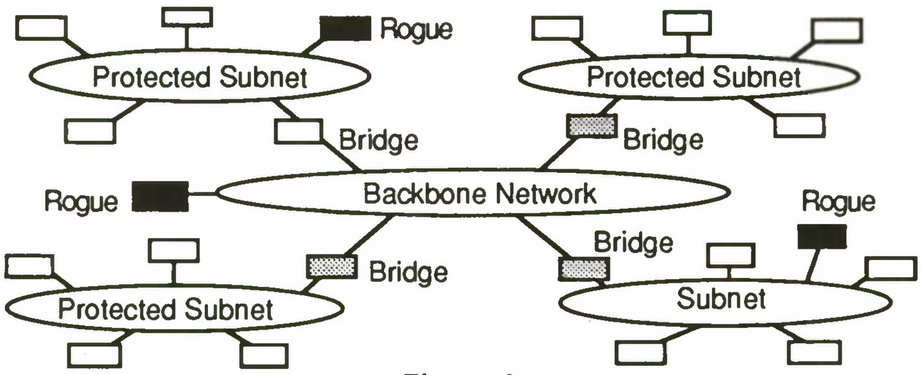


Figure 2

Rogue stations are those that participate in unauthorized activities, whether or not the station is authorized to be attached to the LAN. These rogue stations exploit the risks that have been identified, necessitating the indicated security services. Precautions are necessary to provide protection from these stations wiretapping into the backbone LAN. LAN security services are also necessary to prevent abuse by systems which are authorized to be connected to the LAN, but are being used in an unauthorized fashion. Without the proper security services, even protected subnetworks are susceptible to abuse.

Ultimately, protection of application data can be provided at the application layer. However, in practice, it is desirable to protect information both at the highest possible point in the protocol stack (i.e., the application layer) and any layers at which subnetworks and routing are implemented. This is true for several reasons.

First, security services provided at any layer of a protocol stack, protect only the Service Data Unit (SDU), i.e., the data portion, of that layer's PDU. If data integrity is provided at an upper layer, the header information from that layer and all lower layers is left unprotected. One example of data in a Layer 2 information PDU that is unprotected, even in the presence of higher layer security services, is the security option specified for ISO CLNP, which is included in the U.S. Government Open Systems Interconnection Profile (U.S. GOSIP). Since this data is contained within the Network Layer header, it cannot be protected by security services provided above the Data Link Layer.

Second, PDUs that originate and terminate within Layer 2 are also unprotected in the presence of security services at upper layers. Examples of this type of PDU are the TEST and XID PDUs in ISO 8802-2 LLC, which is also part of the U.S. GOSIP. Network management uses these PDUs, creating a need for protection for this type of PDU as well as information PDUs. ISO 7498-2 considers only information PDUs. It does not address administrative functions and artifacts of protocols. Connectionless data integrity at the Link Layer will provide protection for this type of PDU, as well as information outside the boundary of protection of higher layer security services.

Third, security services provided at the Link Layer provide uniform, common protection for all applications from risks that are intrinsic to LANs and the increased connectivity they provide. Security services provided at another layer can neither take advantage of the attributes of a LAN nor be affected by the deficiencies of a LAN.

Finally, implementations of security at upper layers are developing too slowly to address some users' needs. Emerging LAN security devices can address these needs until upper layer security is available.

CONNECTIONLESS DATA INTEGRITY

ISO 7498-2 defines connectionless data integrity as "the property that the data in a single connectionless PDU has not been altered or destroyed in an unauthorized manner." As the definition indicates, this service inhibits undetected modification of the protected data. This assures the receiving station that the SDU portion of a PDU has not been tampered with since it was transmitted. Given the nature of data transmission and reception at the Link Layer of LANs and the susceptibility of LANs to wiretap, this service is badly needed to protect data on LANs. This service is important not only in its own right, but as a necessary supportive service for authentication services.

Figure 3 illustrates the application of this service to information PDUs. As previously indicated, security services provided at any layer of a protocol stack protect only the SDU portion of that layer's PDU. In implementations where integrity is provided at a higher layer, connectionless data integrity at Layer 2 protects the headers of the layers above the MAC Sublayer up to and including the higher layer at which integrity is provided. The security option specified in the U.S. GOSIP for ISO CLNP is one example of critical data protected in this case. Since this data is contained within the Network Layer Header, it cannot be protected by security services provided above the Link Layer. Modification of the data contained in the security option, combined with the modification of the CLNP header checksum could result in delivery of a PDU to a station not authorized to process that data. In implementations where connectionless data integrity is provided at the Link Layer rather than at a higher layer, application data and all of the headers of the protocol layers above the MAC Sublayer are protected from undetected modification. When implemented at the Data Link Layer, this service also provides protection for logical subnetwork addressing for communities of interest on a common secure backbone LAN.

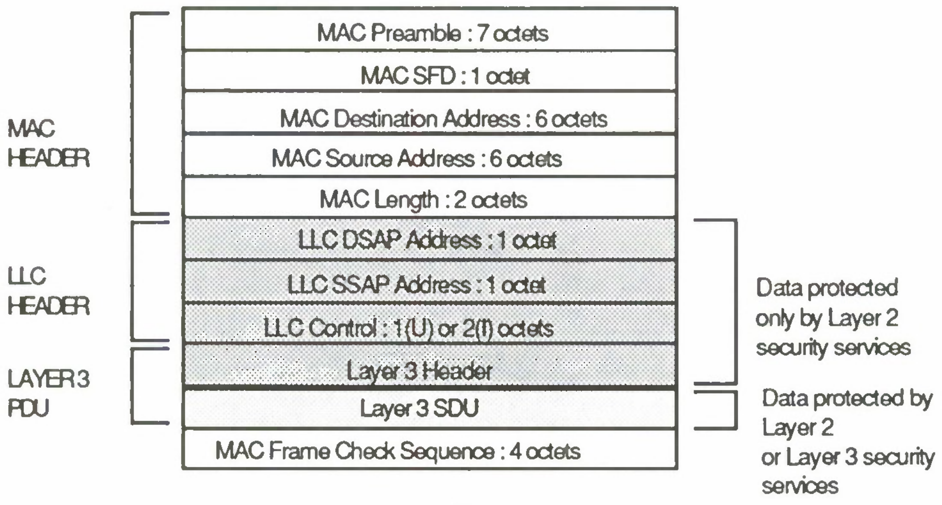

MAC HEADER
LLC HEADER
LAYER 3 PDU
MAC Preamble : 7 octets
MAC SFD : 1 octet
MAC Destination Address : 6 octets
MAC Source Address : 6 octets
MAC Length : 2 octets
LLC DSAP Address : 1 octet
LLC SSAP Address : 1 octet
LLC Control : 1(U) or 2(I) octets
Layer 3 Header
Layer 3 SDU
MAC Frame Check Sequence : 4 octets
Data protected only by Layer 2 security services
Data protected by Layer 2 or Layer 3 security services

Figure 3

Connectionless data integrity is also necessary at the Data Link Layer to inhibit data modification of the data field of the TEST PDU. Figure 4 illustrates the application of connectionless data integrity to this type of PDU. If the data in a TEST PDU is altered by a third party, either during the request or reply phases, it might result in a bad quality path being marked as good. Distortion of TEST data could also cause a good quality path to be marked as bad, but this is indistinguishable from a failure in the media itself and is, in fact, an indication that there is something wrong with the communications path, anyway. This service also protects the integrity of the LLC header fields, preventing misdelivery of the TEST PDU or modification of the Control field, which identifies the PDU as a TEST PDU. Finally, integrity is also necessary as a supportive service for authentication of this type of PDU, since assurance of authenticity of the source address without assurance of the integrity of the source address is of little value.

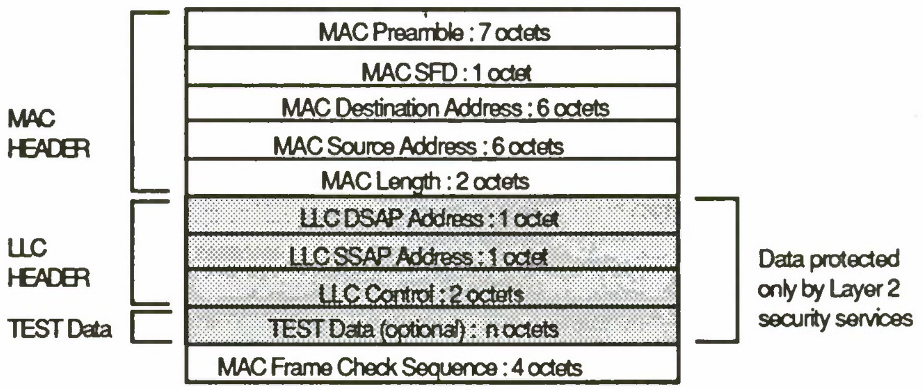

MAC HEADER
LLC HEADER
TEST Data
MAC Preamble : 7 octets
MAC SFD : 1 octet
MAC Destination Address : 6 octets
MAC Source Address : 6 octets
MAC Length : 2 octets
LLC DSAP Address : 1 octet
LLC SSAP Address : 1 octet
LLC Control : 2 octets
TEST Data (optional) : n octets
MAC Frame Check Sequence : 4 octets
Data protected only by Layer 2 security services

Figure 4

DATA ORIGIN AUTHENTICATION

Data origin authentication inhibits one station from masquerading as another to abuse resources attached to a LAN (i.e., unauthorized resource use). This service assures a receiving station that the SDU portion of a PDU came from the station indicated by the Data Link Layer source address in the PDU header. Data integrity is necessary as a supportive service for data origin authentication, since assurance of authenticity of the source address without assurance of the integrity of the source address is of little value. This service protects resources (e.g., file servers) attached to LANs from one station masquerading as another, whether or not the station is authorized to be connected to the LAN. At Layer 2, this service provides protection for logical subnet addressing for communities of interest on a common secure backbone. Given the nature of data transmission and reception at the Link Layer of LANs and the susceptibility of LANs to wiretap, this service is necessary to protect resources on LANs.

Figure 3 illustrates the application of this service to information PDUs at the Data Link Layer. When authentication is provided at an upper layer, the header data from that upper layer and all lower layers, is left unprotected. Again, an example of data in a Layer 2 information PDU that is unprotected even in the presence of higher layer security services, is the security option specified in the U.S. GOSIP for ISO CLNP. Since this data is contained within the Network Layer Header, it cannot be protected by security services provided above the Link Layer. If an unauthorized station masqueraded as an authorized station and replayed the data contained in the security option from a valid PDU, it could result in delivery of data to a station not authorized to process that data. In implementations where data origin authentication is provided at the Link Layer rather than at a higher layer, application data and all of the headers of the protocol layers above the MAC Sublayer are protected. When implemented at the Link Layer, this service also provides protection for logical subnet addressing for communities of interest on a common secure backbone LAN.

Data origin authentication is also necessary at Layer 2 to inhibit modification of the source address field of the source address field of a TEST PDU. Figure 4 illustrates the application of data origin authentication to this type of PDU. If the source address in a TEST PDU is altered, either during the request or reply phases, it might result in a bad quality path being marked as good. Misrepresentation of the source address in a TEST PDU could also cause a good quality path to be marked as bad, but this is indistinguishable from a failure in the media itself and, in fact, is an indication that there is something wrong with the communications path, anyway. Together with the supportive service of integrity, data origin authentication provides necessary protection for this type of PDU, since assurance of authenticity of the source address without assurance of the integrity of the source address is of little value.

ACCESS CONTROL

Access control inhibits unauthorized use of resources. This service is sometimes thought of as a way to inhibit unauthorized disclosure. But, in fact, data confidentiality is used to protect data from unauthorized disclosure. Access control provides assurance that access to a resource is granted only to authorized stations for authorized purposes. Access control can be applied at either the source of a data transmission or at the destination. However, when access control is applied at a PDU's destination, the data has effectively been transmitted to all stations on a LAN before this service is applied. If nothing else, this leaves stations open to unauthorized depletion of network bandwidth and receiver processing resources. Also, due to the manner in which every PDU is effectively transmitted to every station on a LAN and the susceptibility of LANs to wiretap, access control applied at the destination cannot prevent transmissions to stations not authorized to be connected to a LAN. At the Data Link Layer of a LAN, access control, when applied at the source of a data transmission, can inhibit communications between stations not authorized to communicate with one another, including a station authorized to be connected to the LAN and a station not authorized to be connected to the LAN.

Figure 3 illustrates the application of this service to information PDUs. In implementations where authentication is provided at a higher layer, access control at Layer 2 provides protection from abuse of resources that operate upon data contained in the headers of the higher layer at which the service is provided and all other layers above Layer 2. For example, in a network where access control is provided as a Layer 3 end-to-end service over ISO CLNP, PDUs generated on one LAN could be sent to a remote LAN with particular Quality of Service (QOS) option parameters requested and the Record Route option invoked. This would provide information about the intermediate Network Layer systems to a rogue station on the Remote LAN. By also invoking the Partial Source Routing option and limiting the PDU Lifetime, a single station with partial information on the topology of a set of interconnected subnetworks could develop more complete information from Error Report PDUs, without the participation of a second rogue unit. This information could be used to exploit weaknesses in the network, such as identifying operational characteristics of particular routes (e.g., relative levels of congestion, transit delay, or residual error probability). While access control at Layer 2 cannot limit this type of abuse between stations authorized to communicate with one another, it can inhibit this type of communication between stations not authorized to communicate with one another. In implementations where access control is provided at the Link Layer rather than at a higher layer, this service provides protection from abuse of application data and data in the headers of the protocol layers above Layer 2. For example, this service can limit access to a particular file server to only those stations which required that access. It can also prohibit access to a gateway from unauthorized stations.

At the Link Layer of a LAN, this service can prevent use of the TEST PDU from the LLC Sublayer to create an unauthorized communications association. Figure 4 illustrates the application of access control to this type of PDU. Since the data to be used for a TEST PDU is not defined, the entire data field of this PDU could be filled with any data. By transmitting unnecessary TEST PDUs, cooperating stations could transfer any data. While access control will not limit this type of abuse between stations authorized to communicate, it can inhibit this type of communication between stations not authorized to communicate with one another (e.g., a station authorized to be connected to the LAN and a station not authorized to be connected to the LAN).

DATA CONFIDENTIALITY

Data confidentiality inhibits unauthorized disclosure of the protected data. This assures the sending station that the protected portion of a PDU will be available only to the intended recipient. Given the nature of the Link Layer of LANs and the susceptibility of LANs to wiretap, this service is necessary to protect data on LANs. This service is already indicated as appropriate for Layer 2 in ISO 7498-2.

MECHANISMS FOR PROVISION OF SECURITY SERVICES

Concerns that are raised when one suggests expanding the Layer 2 security service profile are: how can the additional security services be provided and what impact will this have on the complexity and performance of the LAN interface to a station. Data confidentiality is most commonly provided via encryption, also referred to as encipherment. In fact, data confidentiality through encryption is what most people associate with network security. While there are other mechanisms for providing data confidentiality, encryption is one of the simplest and most reliable. Fortunately, the mechanism most commonly used to provide data confidentiality, i.e., encryption, can be used to provide all of the indicated security services. In fact, the additional services can be provided with almost no impact to the performance or the complexity of the LAN interface.

Connectionless data integrity is almost an automatic side effect of data confidentiality via encryption. Most cryptographic algorithms produce a checksum or some other mathematical residue which can only be reproduced with the correct combination of cryptographic algorithm, key material, and data. For systems handling classified data, a cryptographic checksum calculated over the data, using an algorithm and key different from those used for the data confidentiality service, might be required. However, this is unnecessary for unclassified data.

Data origin authentication can easily be provided by including a copy of the source address within the encrypted data field, either as a prefix or a suffix to the Layer 2 SDU[2]. As with connectionless data integrity, in systems handling classified data, a cryptographic checksum calculated over the data, using an algorithm and key different from those used for the data confidentiality service, might be required. Again, however, this is unnecessary for unclassified data.

Access control can be effected implicitly through the management and application of cryptographic association, i.e., keying relationships. If all PDUs are encrypted, only those stations with cryptographic mechanisms and knowledge of the correct keying relationships can exchange information. A station without these facilities will be unable to access any of the protected resources.

With the exception of data origin authentication, all of the additional services can be provided as by-products of encryption when used to provide data confidentiality. And data origin authentication can be included so easily, it is hardly worth noting as an exception. Using the single mechanism of encryption, all of the indicated services can be provided with a minimum of impact to the complexity and performance to the LAN interface of an attached station.

SUMMARY

Table 1 summarizes the pertinent attributes of LANs that have been identified, the vulnerabilities that those attributes present, the security threat associated with those vulnerabilities, and the security services required to inhibit exploitation of those risks. In each case, the Link Layer of LANs has been shown to have qualities more like the Network Layer of WANs than those of the Link Layer of WANs. Given these arguments, it makes sense to provide the same range of security services for LANs' Link Layer as WANs' Network Layer.

Table 1

LAN Attribute	Vulnerability	Security Threat	Services Indicated
Data Transmission	Any station can transmit to any other station, using any address	Masquerade, unauthorized resource use	Data origin authentication, access control
Data Reception	Any station can access any transmission	Data modification, unauthorized disclosure	Connectionless data integrity, data confidentiality
Address Space	No implicit controls through address management	Masquerade, unauthorized resource use	Data origin authentication, access control
Geographic Dispersion	Eavesdropping, wiretapping	Data modification, unauthorized disclosure	Connectionless data integrity, data confidentiality

CONCLUSIONS

I have shown which attributes of LANs necessitate security services at Layer 2, the threats associated with those attributes, the services needed to counter those threats, and the results from applying those services. Most of the similarities can be attributed to the facts that subnetworks and routing are functions of the Data

[2] Data origin authentication is assured only to the granularity of the cryptographic key. A key that is unique to the source and destination address pair provides assurance of the individual source host identity; a key shared by a group only provides assurance that the source of the PDU is a member of the group

handling classified data, a cryptographic checksum calculated over the data, using an algorithm and key different from those used for the data confidentiality service, might be required. However, this is unnecessary for unclassified data.

Data origin authentication can easily be provided by including a copy of the source address within the encrypted data field, either as a prefix or a suffix to the Layer 2 SDU[2]. As with connectionless data integrity, in systems handling classified data, a cryptographic checksum calculated over the data, using an algorithm and key different from those used for the data confidentiality service, might be required. Again, however, this is unnecessary for unclassified data.

Access control can be effected implicitly through the management and application of cryptographic association, i.e., keying relationships. If all PDUs are encrypted, only those stations with cryptographic mechanisms and knowledge of the correct keying relationships can exchange information. A station without these facilities will be unable to access any of the protected resources.

With the exception of data origin authentication, all of the additional services can be provided as by-products of encryption when used to provide data confidentiality. And data origin authentication can be included so easily, it is hardly worth noting as an exception. Using the single mechanism of encryption, all of the indicated services can be provided with a minimum of impact to the complexity and performance to the LAN interface of an attached station.

SUMMARY

Table 1 summarizes the pertinent attributes of LANs that have been identified, the vulnerabilities that those attributes present, the security threat associated with those vulnerabilities, and the security services required to inhibit exploitation of those risks. In each case, the Link Layer of LANs has been shown to have qualities more like the Network Layer of WANs than those of the Link Layer of WANs. Given these arguments, it makes sense to provide the same range of security services for LANs' Link Layer as WANs' Network Layer.

Table 1

LAN Attribute	Vulnerability	Security Threat	Services Indicated
Data Transmission	Any station can transmit to any other station, using any address	Masquerade, unauthorized resource use	Data origin authentication, access control
Data Reception	Any station can access any transmission	Data modification, unauthorized disclosure	Connectionless data integrity, data confidentiality
Address Space	No implicit controls through address management	Masquerade, unauthorized resource use	Data origin authentication, access control
Geographic Dispersion	Eavesdropping, wiretapping	Data modification, unauthorized disclosure	Connectionless data integrity, data confidentiality

CONCLUSIONS

I have shown which attributes of LANs necessitate security services at Layer 2, the threats associated with those attributes, the services needed to counter those threats, and the results from applying those services. Most of the similarities can be attributed to the facts that subnetworks and routing are functions of the Data

[2] Data origin authentication is assured only to the granularity of the cryptographic key. A key that is unique to the source and destination address pair provides assurance of the individual source host identity; a key shared by a group only provides assurance that the source of the PDU is a member of the group

Link Layer in LANs and the nature of data transmission and reception at Layer 2 of LANs. The result is a set of arguments for a Layer 2 security service profile that is more extensive than the profile currently defined in ISO 7498-2. This will allow LAN implementations to address pertinent security threats in the layer at which the threats exist, while maintaining compliance to ISO standards.

It is important to emphasize that it is not mandatory to provide these services at Layer 2 in every device which supports ISO standards based systems. In some instances, security services at Layer 2 will be necessary and sufficient to support a particular system. In other cases, Layer 2 security services will be used in conjunction with security services at other layers. In some cases, Layer 2 security services will be unnecessary and inappropriate.

Security services at Layer 2 will address different needs than security services provided at other layers. The ISO Security Architecture should be modified to accommodate these needs.

REFERENCES

ANSI/IEEE Standard 802.2-1985; Local Area Networks: Logical Link Control; Institute of Electrical and Electronic Engineer, Inc.; December 1984

ANSI/IEEE Standard 802.3-1985; Local Area Networks: Carrier Sense Multiple Access with Collision Detection; Institute of Electrical and Electronic Engineer, Inc.; December 1984

ANSI/IEEE Standard 802.4-1985; Local Area Networks: Token-Passing Bus Access Method; Institute of Electrical and Electronic Engineer, Inc.; February 1985

ANSI/IEEE Standard 802.5-1985; Local Area Networks: Token Ring Access Method; Institute of Electrical and Electronic Engineer, Inc.; April 1985

Berson, Thomas A. and Beth, Thomas, editors; Lecture Notes in Computer Science: Local Area Network Security; Springer-Verlag; April 1989

International Standards Organization 7498-2-1988(E) Information Processing Systems--Open Systems Interconnection--Basic Reference Model--Part 2: Security Architecture

International Standards Organization 8473: 1988 (E) Information Processing Systems--Data Communications--Protocol for providing the connectionless-mode network service

International Standards Organization 8802-2-1987 Information Processing Systems--Data Communications--Logical Link Control

P802.10/D5: Standard for Interoperable LAN Security (Draft); IEEE 802.1- LAN Security Working Group, Kirk Barker, editor; July 1989

Tanenbaum, Andrew S.; Computer Networks; Prentice-Hall, Inc.; 1981

United States Government Open Systems Interconnection Profile (U.S. GOSIP), Version 2; Federal Register; National Institute of Standards and Technology; July 1989

ACKNOWLEDGEMENT

I would like to acknowledge the support of the D115 department staff and the INFOSEC Center at MITRE-Bedford, as well as the IEEE 802.10 LAN Security Working Group in the development of this briefing. I would also like to acknowledge the insight and review provided by Steve Kent, of BBN Communications Corporation, and Dave Gomberg, of MITRE-Washington.

Trusted MINIX: A Worked Example

Albert L. Donaldson
ESCOM Corporation
12206 Waples Mill Road
Oakton, VA 22124

John W. Taylor, Jr.
General Electric M&DSO
P.O. Box 8048
Philadelphia, PA 19101

David M. Chizmadia
National Computer Security Center
9800 Savage Road
Fort George G. Meade, MD 20755

ABSTRACT

The Trusted MINIX system is being developed to provide a worked example of C2 security mechanisms and assurances based on MINIX Version 1.5. MINIX is a small UNIX-like operating system for the PC/AT workstation, originally developed by Andrew Tanenbaum as a teaching tool for operating systems classes. Although the computer system will generally be used by only a single user at a time, MINIX was designed for multi-user, multi-tasking operation. From this perspective, the security modifications required for Trusted MINIX are essentially the same as for any multi-user system. However, MINIX was designed with a more modular internal structure than the monolithic UNIX kernel, and this structure affects how security features are added to MINIX. This paper gives an overview of the worked example, both from historical and technical perspectives.

1. Background

Trusted MINIX has its roots in the National Computer Security Center's (NCSC) Rating Maintenance Phase (RAMP)[1] class. A portion of the RAMP class involves security analysis of system changes in order to be sure that the changed system remained consistent with the TCSEC requirements. Early RAMP students attempted to analyze generic changes in a context free environment. Unfortunately, it was was difficult to analyze these changes to the level of detail necessary to provide a useful exercise.

The NCSC concluded that the context of a specific system, with specific changes to that system, were needed to provide a useful class exercise. The difficulty lay in choosing the correct system on which the exercise should be built. A proprietary operating system would not suffice for a multi-vendor class, nor could the NCSC choose a system that would serve as an implicit endorsement of an evaluated system (or system currently in evaluation). The system needed to be conceptually simple enough that students with different operating system backgrounds would be able to grasp the core concepts with little difficulty.

MINIX was chosen as the example system. Unfortunately, standard MINIX does not meet all the requirements of any class of the TCSEC. Therefore, the NCSC embarked on a vigorous campaign to "pretend" that MINIX met the C2 requirements. Auditing and testing magically appeared in discussions about the system and high level design documentation was written. Even though this provided a context upon which to scrutinize system change, the context was internally inconsistent. The students were quick to realize this, and this fact detracted from the benefits gained by the specific context.

A determination was then made by the NCSC to have MINIX built to meet the C2 requirements. However, during the course of creating the class exercise, some interesting discoveries were made: the RAMP class need not be the only beneficiary of a worked example. Since the example system would meet all C2 requirements, it could be used to provide trusted system vendors with examples of TCSEC documentation, such as design documentation and *Trusted Facility Manual* (TFM). If building MINIX also entailed RAMPing MINIX, a *Rating Maintenance Plan* would also be available. These documents, when used in conjunction with the corresponding "Rainbow Series" guide, could

[1] Briefly, RAMP is the process by which vendors maintain their NCSC ratings on subsequent product versions. For more information, refer to the NCSC's *Rating Maintenance Program Document* [1].

provide the vendors of trusted systems with the necessary tools and examples to create documentation meeting the TCSEC requirements. It is anticipated that this may, in fact, speed the evaluation process by increasing the productivity of both vendors and evaluators.

Another application of a worked example is that it can be used for internal training of evaluators without the risks associated with the "trial-by-fire" scheme currently in use. Lastly, the worked example would fill the void in the RAMP class by providing a consistent, evaluated system upon which to perform change security analysis. With these as possible beneficiaries of the worked example, the flavor of the requirements changed slightly. No longer was having the best C2 system features top priority; rather, C2 assurances, in particular, documentation, took a leading role.

In September 1989, the NCSC contracted with ESCOM Corporation to develop the Trusted MINIX operating system and its documentation. The contract called for ESCOM to develop the system for use as a non-proprietary "worked example" of a trusted computer security product. ESCOM's role throughout this contract has been that of a commercial vendor developing a candidate C2 product rather than a contractor developing data items.

2. Technical Overview

Trusted MINIX was developed as a Controlled Access implementation (C2) of MINIX 1.5 for the IBM PC/AT workstation. MINIX [2] is a multi-user, multi-tasking operating system designed to be compatible with UNIX[2] (Version 7) from the user's perspective, but with a more modular internal structure, and with widely available, published source code. The most recent release, Version 1.5, is designed to provide POSIX system call compatibility.

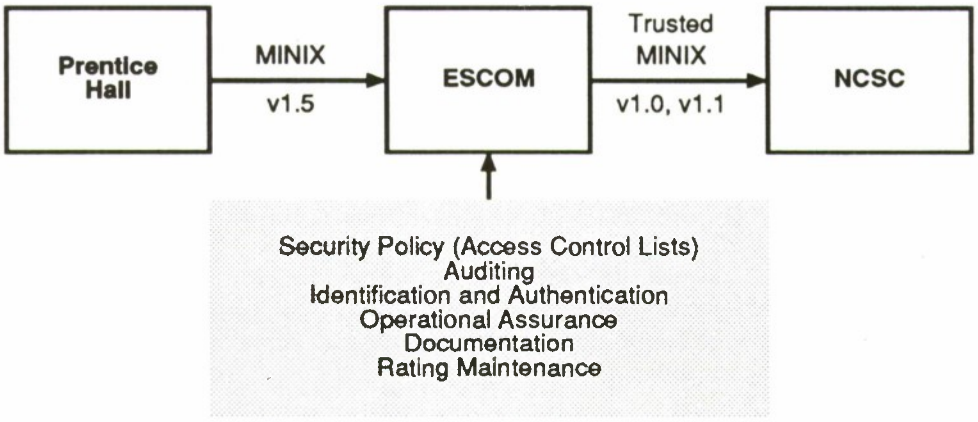


Figure 1. Trusted MINIX Product Concept.

The *Trusted Computer System Evaluation Criteria* (TCSEC) [3] requires C2 systems to include mechanisms to make users individually accountable for their actions through login procedures, auditing, and resource isolation. As shown in Figure 1, Trusted MINIX also provides access control lists (ACLs), a B3 security mechanism. Extensive user, administrator, test, and design documentation have been developed, and a subsequent revision to the system was performed in accordance with the RAMP requirements of the Trusted Product Evaluation Program.

Although Trusted MINIX has been designed specifically for the IBM PC/AT workstation, both MINIX and Trusted MINIX run without software changes on other hardware-compatible systems using the Intel 80286, 80386, and 80486 processors. Trusted MINIX preserves the general architecture of standard MINIX and is source and binary code compatible with most existing MINIX programs.

[2] UNIX is a trademark of AT&T Bell Laboratories.

During this project ESCOM performed several tradeoffs regarding the design of the Trusted MINIX system. In addition to obvious factors such as completing the system within the allocated budget and schedule, primary considerations included:

(1) Succinctly meeting the C2 requirements.

(2) Providing high quality system and user documentation.

(3) Providing a conceptually simple approach in which mechanisms are straightforward, easy to use, and easy to understand. A security mechanism that is not used because it is too cumbersome or is not well understood can actually reduce the security of a system.

(4) Following the original MINIX project goals of modularity, readability, and smallness.

3. Trusted Computing Base

Trusted systems are trusted to protect information – that is, to allow access to data only in accordance with the system's access control policy. Trusted MINIX enforces a discretionary access control (DAC) policy which allows individual users to control access to their data on a "need to know" basis. Trusted MINIX also provides individual accountability by requiring proper identification and authentication of the user before giving access to the system, and by providing the capability for a privileged user to audit security-relevant events within the system. The trust provided by the Trusted MINIX system depends equally upon the proper operation of the DAC mechanisms, individual accountability for system users, and assurances that the system is developed properly.

The parts of the system that collectively provide this trust are referred to the Trusted Computing Base (TCB). As defined by the TCSEC, the TCB is "the totality of protection mechanisms within a computer system -- including hardware, firmware and software -- the combination of which is responsible for enforcing a security policy." The Trusted MINIX TCB includes the system hardware, and firmware, and critical software such as the kernel, device handlers, Memory Manager (MM), File System (FS), and other utilities, commands, and system software. Although modularity is not a requirement for class C2 TCBs, the Trusted MINIX system is internally structured into well-defined and largely independent entities.

Figure 2 shows the overall structure of the Trusted MINIX system. All software in Layers 1 through 3 is included in the TCB, along with certain privileged user programs in Layer 4. Version 1.5 of MINIX includes over 150 user commands, a relatively small number of these that run with superuser privilege or are necessary for system administration are included as part of the TCB.

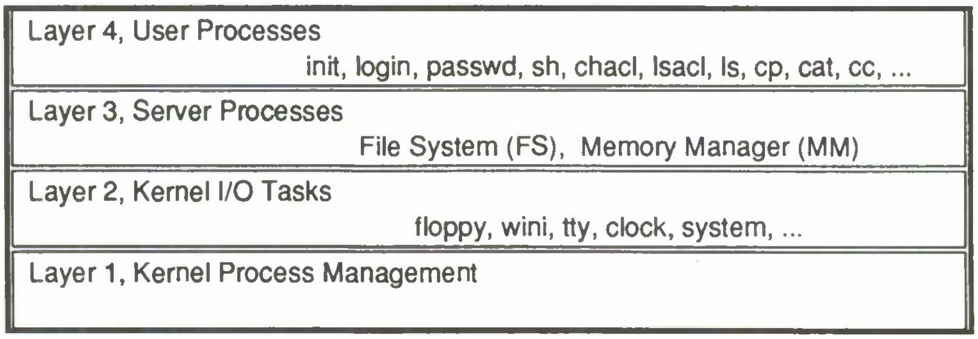

Layer 4, User Processes
init, login, passwd, sh, chacl, lsacl, ls, cp, cat, cc, ...
Layer 3, Server Processes
File System (FS), Memory Manager (MM)
Layer 2, Kernel I/O Tasks
floppy, wini, tty, clock, system, ...
Layer 1, Kernel Process Management

Figure 2. Trusted MINIX Internal Structure.

Each of these four layers is characterized by a distinct hardware privilege and execution priority. User processes at Layer 4 have a low privilege (preventing access to memory segments used by lower layers) and a low priority (they will be run only when the lower layers cannot run). Server processes have increased privilege (allowing servers to access user process memory) and increased priority to ensure that they will run before user processes. The Kernel I/O

Tasks have even higher privilege and priority necessary to manage the physical hardware within the system, and the Kernel Process Management Layer manages the privilege and priority mechanisms for the rest of the system.

Trusted MINIX runs in 286 protected mode and uses the 286 protection ring mechanism to enforce the privileges associated with each layer. In addition, the kernel sets up 286 segment descriptors for each process, thus providing separation of processes (even at the same privilege level).

Although the primary purpose of the TCB is to enforce the system's policy, only a relatively small portion of the TCB (kernel, FS, MM) is directly involved with making access control decisions. This portion of the TCB implements the *reference monitor concept* (TCSEC, Section 6.1), that is, it enforces the authorized access relationships between *subjects* and *objects* of a system.

3.1. Subjects

The TCSEC defines the term "subject" as including all active entities such as persons, processes, or devices that cause information to flow within a system or affect the state of the system. As shown in Table 1, the only subjects defined for the Trusted MINIX system are user processes. All users are eventually represented by one or more processes, usually including the shell interpreter *sh*. As with UNIX, each process is identified by a unique process identifier (PID). User traceability is provided by maintaining effective and real user and group IDs (UID, GID, respectively) for each process.

Table 1. Trusted MINIX Subjects and Objects.

MINIX Entities	Subject	Object	Named Object	Storage Object	System Object	Public Object
User Processes	x	x	-	-	-	-
MINIX Files:						
- Regular Files	-	x	x	x	-	-
- Directories	-	x	x	x	-	-
- Device Special Files	-	x	x	x	-	-
- Named Pipes (FIFOs)	-	x	x	x	-	-
Unnamed Pipes	-	x	-	-	-	-
MINIX Messages	-	x	-	-	x	-
Disk Blocks	-	x	-	x	-	-
Memory Buffers	-	x	-	x	-	-
Registers	-	x	-	x	-	-
System Clock	-	x	-	-	-	x

3.2. Objects

The term "object" is defined by the TCSEC as a passive entity that contains or receives information. As shown in Table 1, there are various types of objects, including named objects, storage objects, system objects, and public objects, with different requirements for protection.

Named objects can be individually addressed, read, and written by arbitrary user subjects. The TCSEC requires the TCB's access control mechanisms to protect all named objects within the system. Named objects within Trusted MINIX include all files within the MINIX file system, including regular files, directories, and devices. These files can be directly manipulated at the TCB interface, may be destructively written by multiple users, and can serve as a channel for information between users.

Storage objects can be read and written by user subjects. The TCSEC requires the TCB to remove residual information from storage objects before allocating them to another subject. For Trusted MINIX, storage objects also consist of MINIX files, directories, and devices.

System objects are protected entities internal to the TCB (for example, firmware, process table, and inode table) that cannot be used for direct communications between subjects. Since they cannot be used to transfer information from one user to another, they do not need to be explicitly addressed by the access control policy. The Trusted MINIX message mechanism allows user processes to communicate with the kernel, MM, and FS processes. This message passing mechanism is a form of inter-process communications (IPC), but it needs no further access control mechanism because user processes are not permitted to send messages directly to other user processes.

Public objects are objects such as the system clock that can be read but not modified by the normal user. Since they cannot be used to transfer information from one subject to another, they do not need to be addressed by the access control policy.

4. Discretionary Access Controls

Because there is only one type of named object (MINIX files) to be considered, the Trusted MINIX DAC design is much less complicated than for other systems. Other systems (such as System V UNIX) allow direct interactions among user processes via IPC or shared memory, but these mechanisms are not available in Trusted MINIX.

The DAC policy for Trusted MINIX is enforced entirely within the FS program based upon an ACL stored in the inode of each file system object. This provides stronger protection against corruption than if the information were stored in a data file or other visible object.

The Trusted MINIX ACL consists of up to eight elements, with each element specifying access permissions for a user ID, group ID, or all other unspecified users on the system. Each element identifies the same permission set (read, write, execute/search) provided by standard MINIX. As shown in the following example, the Trusted MINIX ACL allows access to be specified for multiple users (charlie, lucy, hagar) and groups (peanuts, kudzu):

	Type	UID or GID	Permission
1	USER	charlie	rwx
2	GROUP	kudzu	r-x
3	GROUP	peanuts	r-x
4	OTHERS		--x
5			
6	USER	lucy	r-x
7	USER	hagar	---
8			

Figure 3. Example of Trusted MINIX ACL Structure.

4.1. Compatibility

The predominant consideration for other DAC (TRUSIX, P1003.6) [4,5] working groups has been to provide backwards compatibility with existing software and user interfaces. While such compatibility is important for vendors supporting customers running binary-only applications, it is a relatively low priority for Trusted MINIX.

ACLs and modes represent two distinct discretionary models, and the combination of the two models results in complex interactions that must be understood not only by the developers but also by the users of the system. This makes such a system less effective in enforcing the organization's security policy.

Consequently, ESCOM has implemented a pure ACL approach without mode permissions. This approach has been described as "the cleanest model theoretically because its discretionary control is based on a single, powerful model" [6]. In addition, it is relatively easy to map the existing MINIX system calls that use mode permissions into an ACL approach, and this provides compatibility for existing binary programs.

4.2. ACL Evaluation

Access control is enforced when a process attempts to introduce a file into its address space by opening the file. When this happens, FS will determine whether the requested form of access (for example, read-only, read-write) is to be granted by checking the process' identity (UID and GID) against elements in the file's ACL.

Although ACL elements are not stored in any particular order, they are evaluated in order of most-to-least-specific elements. That is, the ACL will be searched first for matching USER entries, then for matching GROUP entries, then finally for an OTHERS entry. The permissions associated with the first matching entry are used to determine access. If there is no matching element (USER, GROUP, or OTHERS), the requested access is denied. This approach allows access to be explicitly denied for an individual user, even if that user is a member of a group that is allowed access.

4.3. Object Creation

Perhaps the most important issue during object creation is how to set the access control permissions for the new object. Trusted MINIX has replaced the standard *umask* mechanism with an ACL inheritance mechanism where each new object receives its initial ACL from the ACL of the parent directory. As described in other references, this approach is probably the most natural for user and shared project directories, since files inherit permissions from the containing directories.

Using this approach, newly created files inherit an ACL that is derived from the parent directory ACL and the requested permissions from the calling program[3]: The ACL for the new object is copied from the parent directory ACL. An element is created for the owner of the object (if the owner is not already listed on the ACL) with the owner permissions requested by the calling program. All other elements inherited from the parent directory ACL are ANDed with either the group or others bits from the calling program.

This approach is used to initialize the ACL for all newly-created objects. However, certain programs needed to be modified to change this initial ACL to comply with the historical usage of the program. For example, *cpdir -s* duplicates permissions from the origin directory to the destination directory and *tar* provides an option to save and restore ACL information.

5. Object Reuse

The protection philosophy for object reuse is to ensure that the authorizations and contents of reusable objects are properly initialized before the objects are made available to a new user. The TCSEC requires the following mechanisms to be provided for storage objects:

[3] For compatibility with applications using MINIX mode permissions, these are the low-order nine bits of the file mode passed to *creat()*, *mkdir()*, or *mknod()*.

(1) Revoke previous authorizations to the object

(2) Overwrite or clear any residual information remaining in a physical storage location before allowing another user to have access to the object.

The storage objects listed in Table 1 include not only the named objects addressed by the DAC policy (files, directories, etc.), but also lower-level items such as disk blocks, memory buffers, and device registers that may contain residual information. Standard MINIX satisfies the TCSEC C2 requirements for object reuse without any changes other than documenting the mechanisms.

6. Identification and Authentication

The DAC mechanisms described above depend upon the principle of individual accountability. The Trusted MINIX system provides individual accountability by requiring proper identification and authentication of the user before giving access to the system.

6.1. Protecting Encrypted Passwords

Trusted MINIX provides a protected (or "shadow") password file (/etc/tcb/passwd) to prevent general users from being able to read encrypted passwords. The public file (*/etc/passwd*) is still available, but the password field is not used.

6.2. Password Selection

The Trusted MINIX *passwd* program has been modified to filter out certain "weak" passwords, as described below:

(1) Passwords must have a length of at least six characters.

(2) Passwords must contain a non-alphanumeric character (for example, a punctuation mark or a mathematical symbol).

(3) The new password must differ from the previous one.

(4) Trusted MINIX disallows access to the system if the user has a null password. This ensures that the system administrator will set up a password for a new user.

Instead of implementing automatic password aging, Trusted MINIX provides a date field in the protected password file that is changed each time the password is changed. This information can be used by the system administrator to review the current age of users' passwords on the system.

6.3. Login

The Trusted MINIX *login* program authenticates each user's identity before allowing access to the system. It performs the following new functions:

(1) Require a password to be entered, even if the login name is bad. Standard MINIX does not ask for a password if the login name is bad; this allows the user to find valid login names more quickly.

(2) After successful authentication, notify users who login successfully of the date and time of last login and the number of unsuccessful attempts since then. This information is copied from the */etc/tcb/lastlog* file, which maintains information about each user's last login and unsuccessful attempts.

(3) Update the */etc/tcb/lastlog* file with the new port, time, and failure information.

7. Audit

In addition to strengthened identification and authentication mechanisms, Trusted MINIX supports the principle of individual accountability by providing the capability for a privileged user to audit security-relevant events within the system.

7.1. Audit Events

The requirement for auditing at C2 includes: "use of identification and authentication mechanisms, introduction of objects into a user's address space (e.g., file open, program initiation), deletion of objects, actions taken by computer operators and system administrators and/or system security officers, and other security relevant events." Trusted MINIX provides for auditing the following types of events:

Table 2. Auditable Events

Type of Event	Location	I&A	Object	Admin	Other
User Commands					
login	login	x			
su	su	x			
lpr	lpr				x
passwd	passwd				x
System Calls					
fork, exec	MM		x		
open, close	FS		x		
creat, mknod	FS		x		
link, unlink	FS		x		
chroot	FS			x	
stime	FS			x	
chown	FS			x	
mount, umount	FS			x	
audit	FS			x	
kill	MM				x
aclctl	FS				x
setuid, setgid	MM				x
privilege override	FS, MM			x	
failed I/O	FS				x

The second column shows the location where the audit event will be generated, either a trusted user program, server, or the kernel, depending on the type of audit. For example, the *unlink()* system call is implemented within the FS server by the *do_unlink* procedure.

Privilege override allows auditing of events where an operation succeeds only because it is requested by the superuser (UID 0). These situations occur in various system calls, for example, *open()*, *chown()*, *acl()*, *kill()*, and *setuid()*.

7.2. Audit Architecture

As described above, the audit function in Trusted MINIX is necessarily distributed throughout the TCB, with the majority of audits being generated within FS and (to a lesser extent) MM.

Because of this modularity already inherent in the operating system, there were a number of design alternatives for implementation of the audit collection function. The preferred approach was to implement a new audit server at the same level as MM and FS, however, ESCOM implemented the audit collection function within the FS server in order to avoid problems with potential deadlock between FS and an external server. Because the FS and MM servers are "single-threaded" (that is, they only process one transaction at a time), there is the possibility of a synchronization deadlock. Andy Tanenbaum [7] observed that the current MINIX implementation relies on there only being two servers, with limited, one-way (MM-to-FS) interaction between the two servers. Recording audits within FS also makes for optimal performance, since well over half of the system calls are performed within FS. The collection and writing of local audit information within FS consists of a simple procedure call.

7.3. Selective Audit

The TCSEC requires a means to selectively audit the actions of users based on individual identity. This can either be done by pre-selection (audit only the selected users) or post-selection (scan the audit trail for events matching the user). Trusted MINIX allows pre-selection of the types of events to be recorded. The audit reduction tool (*auditfmt*) allows post-selection of audit entries matching a particular user ID, group ID, or inode. *auditfmt* is designed to operate as a filter on an audit file (not necessarily the current audit file), and send to standard output audit records matching the specified criteria.

One of the areas where Trusted MINIX differs from other audit implementations is that the individual instrumentation points send all audit events to FS, even if the event will eventually be discarded. This centralized approach to audit selection has some minor performance implications, but is more modular, easier to modify, and conceptually cleaner than distributing the decisions to each of the instrumentation points.

8. Documentation

Perhaps the most useful aspect of Trusted MINIX is the example documentation. All documentation was written with the assumption that the reader has a user level understanding of standard MINIX. The documentation can be broken into four subgroups: design, test, user, and RAMP.

8.1. Design Documentation

As shown in Figure 4, six documents were written to satisfy the TCSEC design documentation requirements. This is in addition to previously existing documents that were also used to satisfy the requirements.

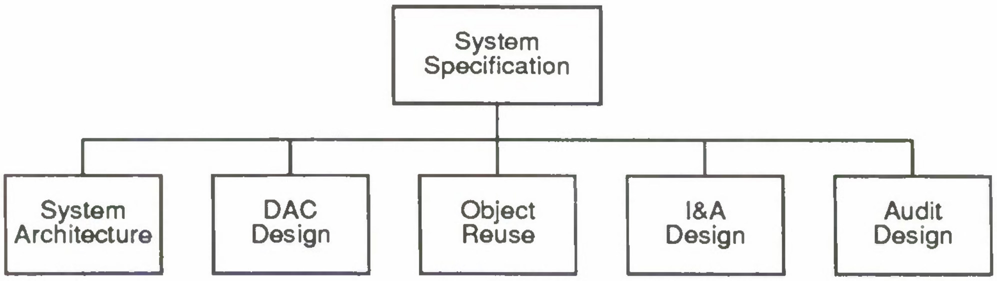


Figure 4. Trusted MINIX Design Documentation

The first document is the *System Requirements Specification*, which identifies the requirements for security features and assurances built into Trusted MINIX system, and describes the relationships among the features. There are four subsystem design documents, which discuss in more detail the functional requirements, the design of the system, and implementation specifics. Unfortunately, these documents did not adequately cover the assurance requirements of the TCSEC, nor did they discuss how the security features fit within the Trusted MINIX system components (such as the kernel, file system, and memory manager). Therefore, the *System Architecture* design document was written to cover these specific issues using a structured breakdown of the Trusted MINIX system. Additional commercially available documentation has been identified for use in documenting the hardware design and implementation.

8.2. Test Documentation

The test documentation consists of a single document covering the design of the Trusted MINIX security relevant tests, the expected results from those tests, and the actual results of the tests.

8.3. User Documentation

The Trusted MINIX user documentation consists of a *Trusted Facility Manual* and a *Security Features Users Guide*. The TFM discusses the issues associated with installing and administering a Trusted MINIX system. It discusses the use of the trusted administrator shell, the use of the auditing mechanism, and various other administration functions and details. The SFUG leads a user through the logon process and explains in detail the security features provided by the system, as well as the users role in system security. Both documents contain manual pages for the security features referenced in the body of each. The SFUG also includes the remaining manual pages not related to security.

8.4. RAMP Documentation

The RAMP documentation consists of all that is needed for a single cycle of Rating Maintenance. This includes a *Configuration Management Plan* and *Rating Maintenance Plan*. The *Configuration Management Plan* takes the view of managing change, as opposed to the *Rating Maintenance Plan*, which takes the view of managing releases. Other RAMP documentation includes the configuration management evidence necessary to support RAMP, as well as a *Rating Maintenance Report* and supporting documentation.

9. Conclusions

The work done by ESCOM to develop the Trusted MINIX system and its associated documentation represents only one side of the effort required to developed a complete worked example, since it only covers the evaluation process from the product developer's side. The other side of the worked example involves the evaluation of the Trusted MINIX system and the development of the documentation associated with that evaluation. This documentation includes the *Preliminary Technical Report* (PTR), the *Initial Product Assessment Report* (IPAR), the *Evaluation Test Plan*, and the *Final Evaluation Report* (FER). The PTR is a cursory analysis of the proposed system (either a design or an existing untrusted base) to determine how feasible it is to complete a trusted product evaluation. The IPAR is based on a detailed technical analysis of the developer's design and user documentation and describes how the system satisfies the requirements of the TCSEC. The IPAR is both the blueprint for the actual product evaluation and the basis of the FER. During the evaluation, the team prepares and runs security tests beyond those done by the developer, these test are documented in the Evaluation Test Plan. Finally, the team produces a report for public release that describes how the product satisfies the TCSEC requirements.

During the development of Trusted MINIX, a team of NCSC evaluators worked with ESCOM to define the security issues and identify possible solutions, in the same way the NCSC usually works with trusted product developers. At the completion of the contract, this team will produce the IPAR, ETR, and FER. At this point, the worked example will be complete with respect to the product evaluation process. The final aspect of the project will be validating the evidence developed under the contract to show the the system maintained its trustedness as it evolved from release 1.0 to 1.1. This will provide the RAMP element of the worked example. When the worked example is complete, the

NCSC will publish it as a series of 11 documents as part of the technical guidelines program (aka the Rainbow Series).

It is expected that the Trusted MINIX will bring the following tangible benefits to the Information Security community:

(1) By providing an example of what the NCSC is looking for in terms of design and user documentation, it will allow product developers to better determine the level-of-effort required to complete the evaluation.

(2) It will provide the basis for quicker and more effective evaluator training.

(3) It will provide a preliminary validation mechanism for fine-tuning the RAMP requirements.

(4) It will provide the consistent example needed to effectively train Vendor Security Analysts, and

(5) It will provide a low-cost and possibly "fun" means for anyone interested in information security to experiment with basic trust technology, both at home and at school.

The use of Trusted MINIX as a worked example doesn't stop here. Work is already underway to use Trusted MINIX as the basis for three follow-on efforts: a B-level worked example; integration into a networked/distributed computing environment; and as a baseline for a trusted system portability study. As we and others gain more experience with Trusted MINIX, we expect to find even more ways to use it to advance the state-of-the-art in Information Security.

10. Acknowledgements

The authors wish to acknowledge the work performed by Brian Beattie, John Hare, Tom Welsh, and Karl Nyberg on this contract. Discussions with trusted UNIX vendors, particularly Michael McChesney of SecureWare and Tim Ehrsam of Addamax, were helpful to ESCOM in formulating initial concepts for this system. Finally, ESCOM wishes to acknowledge the guidance and assistance within the NCSC and the Trusted MINIX evaluation team, particularly the assistance by James Goldston before the contract was awarded.

REFERENCES

(1) *Rating Maintenance Program Document*, National Computer Security Center, NCSC-TG-013.

(2) *Operating Systems Design and Implementation*, Andrew S. Tanenbaum, Prentice-Hall.

(3) *Department of Defense Trusted Computer System Evaluation Criteria*, National Computer Security Center, December 1985.

(4) *Rationale for Selection of Access Control List (ACL) Features for the UNIX System*, National Computer Security Center, 18 August 1989.

(5) *Portable Operating System Interface for Computer Environments, Trusted System Extensions*, Draft 3.

(6) *On Incorporating Access Control Lists into the UNIX Operating System*, "Proceedings UNIX Security Workshop," 1988, Steven M. Kramer, SecureWare, Inc.

(7) Personal communication with Andy Tanenbaum, 5 February 1990.

SECURITY FOR REAL-TIME SYSTEMS

Keith P. Loepere, Franklin D. Reynolds,
E. Douglas Jensen
Alpha Research Group
Concurrent Computer Corporation
1 Technology Way
Westford, MA 01886

Teresa F. Lunt
Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

Abstract

This paper discusses the issues that arise when multilevel security is applied to real-time systems. The Alpha real-time distributed system is used as a means of illustrating these issues. Among the issues discussed are mandatory security, integrity, and denial of service. In addition, it is observed that in real-time systems it may be necessary to make critical trade-offs between timeliness and security. Some approaches to address these issues are proposed.

Introduction

The last several years have seen a flurry of activity in the study, design and application of real-time systems. In the context of this paper, a *real-time* system is one in which the physical resources of the system (most notably processor time) can be precisely controlled.

Real-time systems are used over a considerable range, from low-level sampled-data monitoring and control of physical processes, up to large scale adaptive distributed systems that control multiple low-level real-time systems. It is with this latter class of systems that this paper is concerned. Much of the impetus (and funding) for the development of such systems has come from

This work was supported, in part, by a contract from the U.S. Air Force, Rome Air Development Center.

K. P. Loepere and F. D. Reynolds are currently with the Research Institute of the Open Software Foundation, 11 Cambridge Center, Cambridge, MA.

the Defense community, and there are many proposed applications of these real-time systems to critical Defense problems such as battle management for surface ships, mission management for fighters and strategic planning for SDI. As the real-time market matures, these systems will also find themselves placed into commercial settings, such as the overall control of entire factory processes and the control of financial markets.

Some of these applications raise questions of multilevel security (i.e., separation of information and users based on classification and clearance). For example:

- *process control*: The properties of some materials (hazardous materials, in particular) are categorized by a need-to-know classification by the individual suppliers of the materials. A process control system may handle multiple materials from multiple suppliers where the individual suppliers provide the material properties.
- *battle management*: Mission data is of a very high sensitivity. This data is present in the same (possibly distributed) system as is relatively low sensitivity maintenance data.
- *financial markets*: Company revenue projections are of high sensitivity. Employee salaries and stock purchases are confidential. Employees and executives have clearly differing access.

Thus far, little or no consideration has been given to the computer security implications of real-time systems. The aim here is to call attention to the range of multilevel security issues raised by real-time systems, to suggest a framework in which these issues may be considered, and to propose a research agenda appropriate to the problem. To provide a focus, the study will examine how multilevel security could be integrated into the Alpha real-time operating system under development at Concurrent Computer Corporation. In addition, the concentration will be on mandatory security because it is felt that it is the most pressing security concern.

The requirement to provide secure operation as well as real-time (predictable and controllable) operation interact in a variety of ways.

- All of the circumstances under which the security mechanisms will become active must be controllable and predictable by the applications. The resources consumed by the various security mechanisms (both in time and space) must be controllable and predictable. Although this property of real-time systems largely serves to constrain the manner in which security mechanisms are implemented, it may well affect the semantics of the security services provided.
- Traditional secure systems *virtualize* resources so as to reduce or remove the potential for covert channels. However, proper sharing and control of *physical* resources is vital to accomplishing the mission of a real-time system. As such, an adaptive view must be taken to assure the realization of the system's real-time goals (physical resource control) as well as its security goals (hiding the effects of multilevel resource sharing).
- Meeting real-time goals involves completing the collection of activities that results in the highest aggregate value to the system, where the value of completing an activity varies with time. There is no a priori relationship between the importance or urgency of an activity and its associated access class. "Background" activities such as audit trail generation and analysis may need to take a back seat to more urgent activities at any given time.

- A single, simple security model may not be sufficient for these large, distributed environments. The ability to support application-specific policies that can make trade-offs between critical security and timeliness requirements may be needed.
- Because the real-time systems being discussed are distributed systems in which an application spans multiple nodes, with each subject in the system accessing resources on multiple nodes, the security model must be capable of modeling or accounting for this distribution.
- The real-time systems of concern must be survivable in the face of certain threats. This implies corresponding requirements for system integrity and data integrity.

This paper is organized as follows. The next section provides some background information on real-time computing. This is followed with a description of the Alpha real-time distributed system given in sufficient detail so that the reader can evaluate the proposed approaches that follow. In the section entitled "Security for Alpha" the security issues that arise in Alpha are discussed. Many of these issues are generic to the entire class of real-time systems. In that section some approaches to address these issues are presented. Finally, the paper ends with our conclusions.

Real-Time Computing

A *real-time* system [6] is distinguished from a non-real-time system in that the correctness of its computations depends not only on the values of its outputs but also on the time at which those outputs are produced. Producing the otherwise correct results either too early or too late results in decreased value to the system, possibly jeopardizing the mission of the system, or human life or property. Although any system can be viewed as "real-time" if the hardware is fast enough to always produce results in time, in this paper a system is considered as real-time only if applications can assert their real-time needs and the system manages the resources of the system (often, very precisely) in such a way as to meet those real-time needs.

When applied in the traditional way, the constraint of real-time has the result of producing a system whose most distinguishing characteristic is rigidly deterministic behavior. The spectrum of traditional real-time systems ranges from rudimentary rate-monotonic dispatchers that have been employed in avionics to full-functionality operating systems. Along this spectrum there are differences in the division of resource management responsibility between the system and the application software and in the number and complexity of resources managed at runtime. However, these systems share the objective of maximally deterministic behavior and the approach of employing maximally deterministic techniques. For example, they plan for an anticipated system usage pattern and pre-allocate resources in an attempt to eliminate a priori all variabilities and exceptions.

Our research is concerned with large, integrated, distributed real-time systems. The integrated system is typically comprised of low-level sampled-data subsystems, human-machine interface subsystems, and interconnections to other systems. The integrated system's overall behavior is dynamic and non-deterministic in as much as that its tasks are predominantly aperiodic and asynchronous. Stochastic run-time resource demands and conflicts (fluctuations in load and resource contention, mechanical tolerances in sensors and actuators, and faults, errors, and failures) are inevitable. Most aperiodic as well as periodic tasks have critical time constraints: urgency in time and relative importance in functionality.

The metric of performance is not the speed with which activities can be started. The critical criterion is that the set of activities that results in the highest aggregate value to the system is completed (where value is time-varying), despite dynamic resource demands and conflicts, processing overloads, and hardware or software faults.

The characteristics of the systems intended to perform the mission-critical integration and operation of large, complex, distributed real-time systems such as SDI BM/C^3 differ substantively not only from those of common non-real-time systems such as network connected personal workstations and throughput-oriented super-computers, but also from the traditional small, simple real-time subsystems for low-level sampled data monitoring and control. The differences are manifest primarily in five areas of operating system requirements.

- *Real-time*: meeting as many as possible of the most important aperiodic as well as periodic time constraints, despite dynamic and stochastic runtime resource contention, overloads, and faults
- *Distribution*: managing, in a decentralized fashion, the resources of multiple physically dispersed computing nodes towards the execution of large, complex, integrated, distributed computations to perform a mission
- *Survivability/Integrity*: preserving the mission, human life, and property in a hostile environment with limited or no repairs or downtime during missions of up to decades long
- *Adaptability*: serving a wide variety of applications, each of whose requirements evolve continuously over a lifetime of decades, on a dynamic technology base
- *Security*: preventing unauthorized access or disclosure of mission sensitive data whose characteristics vary over time to sets of individuals or their agents which also vary over time.

A system that satisfies these requirements by necessity has a philosophy that differs from that of traditional systems in several ways.

- *Determinism*: The operating system should behave as deterministically as the application requires, should present deterministic abstractions such as periodic rate-monotonic scheduling to the application user if desired, but should utilize non-deterministic means to achieve those ends most effectively.
- *Exceptions*: The performance of the system must be optimized for the most important cases, which are often high-stress exceptions, such as emergencies due to hostile attack or faults, rather than for the normal, frequent but uneventful cases. The operating system and the application must be designed to anticipate runtime exceptions and to handle them so as to provide the strongest possible realistic assurances about meeting time constraints.
- *Best-Effort Resource Management*: Guarantees of response are not only impossible in general, but, more importantly, honoring guarantees may prevent the system from responding to more critical dynamic demands. Instead, resource management must be performed on a "best effort" basis. The system must get the best results it can within the time constraints, with the available resources. The system should provide runtime predictions of its ability to meet these demands. When not all time constraints can be met, application-specified recourse must be taken, such as gracefully

degrading to minimize the number of missed time constraints, or meeting as many as possible of the most important time constraints.

Alpha

The kernel of the Alpha operating system [13] provides a new programming model that is well suited to writing real-time distributed software. Its principle abstractions are:

- *objects* (passive abstract data types—code plus data), in which there may be any number of concurrent control points
- *method invocation* (similar to procedure calling)
- *threads* (loci of control point execution) which move among objects via method invocation.

These abstractions form the heart of a highly effective real-time distributed programming model. In addition, Alpha provides transaction mechanisms to achieve the necessary consistency of replicated and partitioned data and correctness of distributed execution.

Objects and Classes

An *object* in Alpha is an instance of an abstract data type, created via invocations upon a class object. Each instance of an Alpha client-level (non-kernel) object has a private address space that contains the code and data that make up that object. The kernel considers the universe of objects to be flat. An instance of an Alpha object exists entirely on a single node. Instances can be dynamically migrated among nodes; initial instance placement is specified by the user. Also, objects may be replicated transparently. Alpha objects are intended to normally be of moderate number and size—e.g., 100 to 10,000 lines of code. Everything appears as an object to the programmer: devices, files, etc.

Object naming in Alpha is based on *capabilities* [7]. Alpha's capabilities have the characteristic that they are globally unique over time and are network location independent. An object possesses a set of references to other objects in the form of capabilities (its "C-list"). The capabilities are actually stored within the kernel; a thread executing within an object references them via object local identifiers. Since the actual names of objects are maintained only within the kernel, threads cannot guess the identity of objects. An object has a client visible name only if the creator of an object can "install" a capability to the new object in some existing object. The sensitivity of this existing object determines the sensitivity of the "name" so installed for the new object.

Threads and Thread Segments

An Alpha *thread* is a continuous distributed execution point which transparently and reliably spans physical nodes, carrying its identity, its local state and attributes for timeliness, robustness, etc. These attributes are used by Alpha at each node to perform resource management on a system-wide basis in the best interests (i.e., to meet the time constraints) of the entire distributed application.

There is a single system-wide name-space for threads. Threads are named by capabilities generated when the threads are created.

A thread becomes threaded in an object by invoking a method of that object. It becomes un-threaded by returning from that invocation. At any given time, a thread is executing within one and only one object, with a stack history being maintained by the kernel of the objects in which the thread is currently threaded.

That portion of a thread present within an object in which a thread is threaded is called a *thread segment*. A thread segment possesses some private data not visible to other threads executing within the object. This data consists of pure data (such as a machine stack), as well as a set of capabilities local to that thread segment (its "C-list"). These capabilities are stored in the kernel and are referenced via thread segment local identifiers.

From the point of view of modeling security, the thread segments are the *subjects* in the system.

Capabilities

A *capability* is a reference to an Alpha object, maintained by the kernel, and referenced by threads via object or thread segment local identifiers. A capability contains all of the information necessary to reference the object it names.

There are three uses to which a capability may be put. The primary use of a capability is to invoke a method upon the object to which it refers ("invoke the capability"). A capability may also be used as the target of a thread creation. This operation is effectively the same as an invocation, in as much as that the target of the thread creation is a method invocation of the object to which the target capability refers, but a new thread is created (which will run asynchronously with the creating thread). The third use of a capability is to pass it as an argument to an invocation or thread creation, or to return it as a result of an invocation. It is also possible for a subject to "install" one of its private capabilities into its executing object's "C-list", as well as to make a subject private copy of one of the subject's executing object's capabilities.

Capabilities contain access attributes that restrict (through their absence) the uses to which a capability may be put. Once removed from a capability, an access attribute can not be restored. The initial set of access attributes present in a capability are those present in the capability used to generate the new capability.

Invocation and Thread Creation

The *invocation* of a method of an object is the vehicle for all interactions in the system, including operating system calls. Invocation has synchronous request/reply semantics (similar to RPC); method invocations are block structured. The effect of an asynchronous invocation can be obtained via thread creation.

Invocation masks the effects of physical distribution. Remote objects and object migration provide location transparency. Communication errors are handled by underlying reliable message protocols. The detection and elimination of orphaned computations mask node failures.

Invocations may fail for various reasons, such as protection violation, bad parameters, node failure, machine exception, time constraint expiration and transaction abort. The kernel provides mechanisms for block-structured exception handling to allow the object programmer to designate application-specific handlers for each type of method failure, on a per-invocation basis if desired.

When a capability is invoked, the invoking subject (thread segment) is suspended and a new subject is (effectively) created that is executing within the object to which the invoked capability refers. The invoking subject provides the initial private data and capabilities for the new subject.

A thread creation is identical to a method invocation except that the method is invoked asynchronously, and therefore does not return (any return values are discarded).

The new subject executes the procedure specified by the method invoked upon the object. When the procedure is completed, the subject is (effectively) deleted. In the case of method invocation, the invoked subject may return pure data and capabilities to the invoking subject.

Policy/Mechanism Separation

Alpha strictly adheres to the philosophy of the separation of *policy* and *mechanism*. It has a kernel of primitive mechanisms from which all else is constructed according to a wide possible range of application-specific policies to meet particular functionality, performance, and cost objectives.

Alpha's kernel mechanisms are intended to provide the lowest meaningful level of functionality for an application. Any lesser functionality would result in recurring, inconsistent, inefficient implementation of the desired functionality in the applications. Any greater functionality would limit policy flexibility. Policy modules written at Alpha's system and user layers employ the Alpha kernel mechanisms.

Security for Alpha

As a truly distributed system, Alpha would be described using the "Single Trusted System View" of the *Trusted Network Interpretation* [12]. The interconnection between Alpha nodes in a single Alpha system is considered a part of the kernel and is completely controlled by the kernel. Non-Alpha communications traffic cannot use the Alpha interconnect. There is no object-visible notion of "sessions" or "connections," or "datagrams" within an Alpha system. The interconnect should be viewed as being analogous to a backplane in a multiprocessor system.

The kernel executes in its own hardware protected space. Also, the client objects that form the remainder of the Network Trusted Computing Base (NTCB) are protected from one another and from non-NTCB objects via their separate, kernel-provided address spaces. The NTCB is structured into separate object managers in the kernel and into separate client objects outside of the kernel. A simple protection mechanism, object address space separation and object invocation, provides this structuring. The use of client objects for the structuring of the NTCB allows for excluding non-protection-critical modules from the NTCB.

Access Classes

A security classification, or *access class*, consists of a hierarchical sensitivity level (e.g., TOP-SECRET, SECRET, CONFIDENTIAL, UNCLASSIFIED, etc.) and a set of non-hierarchical categories. The sensitivity levels are linearly ordered. The categories do not have such a linear ordering. However, the set of access classes (⟨sensitivity level, category set⟩ pairs) is partially ordered and forms a lattice [4]. The partial ordering relation is called the *dominance* relation. Access class A dominates access class B if the sensitivity level of A is greater or equal to the sensitivity level of B and the security categories of A include all those of B. For convenience, $A \geq B$ is written to mean that A dominates B.

The access class may further consist of a secrecy component, an integrity component, or both. The *secrecy* component could be a secrecy level, secrecy category, or pair ⟨secrecy level, secrecy category⟩, where *secrecy level* is TOP-SECRET, SECRET, CONFIDENTIAL, UNCLASSIFIED, etc. and *secrecy category* is a set consisting of formal compartments (e.g., CRYPTO). Similarly, the *integrity* component could be an integrity level, integrity category, or pair ⟨integrity level, integrity category⟩ such as introduced by Biba [2]. The lattice on the access classes is defined as the Cartesian product of lattices on the individual components. Note that when integrity is integrated with secrecy, integrity levels are ordered in reverse so that $L_1 > L_2$ means that access class L_1 has a higher secrecy component but lower integrity component than access class L_2. This is because a user is permitted to read down in secrecy but up in integrity, and write up in secrecy but down in integrity.

Subjects and Objects

An Alpha *object* forms a protection domain. Objects have a single access class. That is, Alpha objects have a single secrecy class (i.e., classification) and a single integrity class. Alpha objects never change their classification.

An Alpha *subject* is a thread segment. The subject's access class never changes. In order for a subject to invoke a method of an object (that is, for values to be returned from the invocation), the subject's access class must dominate the access class of the object. A subject may create a new thread (and thereby a subject) in an object whose access class it does not dominate in the case where it does not require return values.

Since the thread is the sole point of control for actions requested by the thread, the thread is the proper entity for which auditing is to be performed.

In Alpha all actions requested by a thread are performed by that thread. As opposed to client/server systems, there is no issue as to the identity of the subject performing actions on remote nodes. The thread is the single entity whose identity need be authenticated and tracked.

Capabilities for Mandatory Security

Objects in Alpha are placed in separate, hardware-restricted address spaces. Access to these objects is completely controlled via kernel-protected capabilities. In order to invoke a method of an object, the subject must own a capability to the desired target object, and the capability must permit the invocation of the desired method. Implemented by the kernel, this single mechanism provides the basis to control sharing of objects.

A capability names an object. The method desired is passed as a parameter. Thus the granularity of access control is at the level of objects.

Both objects and subjects have capability lists ("C-lists"). A subject executing within some object can use or pass as parameters the capabilities in the object's "C-list", as well as those in its own "C-list", if the capability access attributes allow it.

From the point of view of discretionary access, the fact that a subject has a capability means that it can use it—no access decision needs to be made when the subject actually accesses the object—the policy decision was made when the capability was given. The issue is to ensure that the correct mandatory decisions are made when a capability is granted and that any restrictions imposed by the mandatory policy are reflected by and enforced by the capability.

To ensure that mandatory security is satisfied, the following properties are needed:

Simple Security Property: An object or subject X_1 with access class L_1 can hold a capability that permits reading object X_2 which has access class L_2 only if $L_1 \geq L_2$.

**-property*: An object or subject X_1 with access class L_1 can hold a capability that permits writing object X_2 which has access class L_2 only if $L_2 \geq L_1$.

Object Creation Property: A subject X_1 with access class L_1 can create an object X_2 with access class L_2 only if $L_2 \geq L_1$.

Object Deletion Property: A subject X_1 with access class L_1 can delete an object X_2 with access class L_2 only if $L_2 \geq L_1$.

Subject Creation Property: A subject X_3 spawned (either through method invocation or thread creation) by subject X_1 with access class L_1 in object X_2 which has access class L_2 will have an access class L_3 = MAX(L_1,L_2) (L_1 and L_2 must be comparable). If $L_2 > L_1$, no return value can be given to X_1.

The subject creation property and the *-property combined require that a subject can modify its executing object only if the access class of the subject matches that of the executing object.

Alpha's capability mechanism could be used for multilevel security simply by having the kernel perform a mandatory access check whenever an invocation is performed. This could be quite expensive, and very well excessive for a real-time system.

Note that the relevant access decisions to be made depend not so much on the absolute value of the access classes of subjects and objects, but on the result of comparing these values. The results of these access class comparisons can be recorded in capabilities. For this purpose, it is necessary that capabilities have attributes denoting whether they can be used for read or write access—the result of access class comparisons associated with the simple and *-properties. The lack of the write attribute means that use of the capability will not allow the internal state of the object to be changed, but results may be returned. The lack of the read attribute means that use of the capability may allow the internal state of the object to be changed, but results cannot be returned and there can be no indication of success or failure. With these attributes, the above security properties can be amended as follows:

**-property'*: A subject X_1 executing within object X_2 in which X_2 is not writable (that is, X_2 is potentially of lower access class than X_1) can use X_2's read-write capabilities only as read-only capabilities and cannot use X_2's write-only capabilities.

Object Creation Property': If a subject X_1 with access class L_1 creates an object X_2 with access class $L_2 > L_1$, X_1 can receive only write-only capabilities for X_2.

Object Deletion Property': A subject X_1 can delete an object X_2 only if X_2 is writable (that is, X_2 is known not to be of lower access class than X_1).

Subject Creation Property': When a subject X_3 is spawned (either through method invocation or thread creation) by subject X_1 in object X_2 in the case where X_2 is not readable (that is, X_2 is potentially of higher access class than X_1), any capabilities passed by X_1 to X_3 become read-only capabilities in X_3 and no return value can be given to X_1.

These additional constraints expressed in terms of restricting read and write attributes provide the result that objects and subjects cannot hold capabilities that violate mandatory security and that the results of mandatory access decisions are reflected purely in the read and write attributes.

Capabilities, via these attributes, are flexible enough to be used to enforce both simple (traditional) policies, as well as additional policies, such as "close hold". In keeping with Alpha's policy/mechanism separation, such policies could be defined in a module outside the Alpha kernel but within the NTCB. These modules would make use of the basic capability mechanism provided by the Alpha kernel. Multiple policy modules can be defined, so that it is conceivable that multiple such security policies could be in effect in a single Alpha system, supporting different applications. All such policies would be enforced using the Alpha kernel mechanisms.

These mechanisms are used to create and access objects of differing access classes roughly as follows. A subject normally creates objects of its own access class. The subject can install these capabilities into its executing object only if the object is of its own access class, so the result is a set of objects of some access class with capabilities to other objects of that same access class. A subject can also create objects of a higher access class than itself. The subject is then given a capability to this object that lacks read access. This new object cannot obtain writable capabilities to pre-existing objects of its access class since any capabilities passed in via the (only) non-readable capability lose write access in the process. A thread can be created to execute at this higher access class, and that thread can create new objects at that higher access class. The object can receive non-writable capabilities to lower access class objects via arguments passed from a lower access class subject that creates a thread in this higher access class object.

Note that with these attributes deletion of higher class objects is not a covert channel in Alpha as it would be in many systems [8]. If an object is of higher access class than some subject, that subject will only have access to non-readable capabilities through which it can create new threads, but not do normal invocations. If the target object is deleted, that subject still has these capabilities (to a non-existent object), with the fact that the (old) object was of a higher access class than the capability holder still recorded in the lack of the read attribute. The subject can still "create" new threads in this target object which will silently do nothing.

It is interesting to consider what it means for an object to be considered as "labeled" with a particular access class. If a subject creates a new object in such a way as to be given a non-readable capability to that object, then that object can be considered to be at a (potentially unnamed) access class higher than that of the creating subject. Because the capability to the object is non-readable, this new object can only obtain non-writable capabilities to any object at the subject's access class, so it satisfies the mandatory policy rules for an object of higher class. Only the subject need here know what is the "true" access class of this new object. This new object cannot obtain capabilities that are both readable and writable to any existing object, so it does not matter what access class it "really" has; it is sufficient to know that it is higher than that of the creating subject. In this way, subjects can effectively create their own access classes, simply by creating a non-readable object, and allowing that object to create other objects at its (new) access class.

With this understanding, it can now be seen why these mechanisms are appropriate for the domain of real-time systems being described. Not only is the cost of the *mechanisms* that enforce mandatory access made small, but their use is predictable in terms of the exact points at which access decisions are made. Also, by removing the interpretation of mandatory security *policy* from the kernel, the goal of permitting change in the mandatory policy over the long life of these systems (without reconfiguring the system) is achieved.

Integrity and Denial of Service

Integrity can mean many things in a computer system. In this section, the issues which the security community normally refers to by the term "integrity" are discussed. The Alpha kernel provides many mechanisms that pertain to maintaining integrity of data; these mechanisms are beyond the scope of this paper.

Integrity has been used to mean a mandatory policy [2]. Mandatory integrity is similar to mandatory security, in that it enforces the two rules:

- A subject S can read an object O only if the integrity class of the object dominates that of the subject.
- A subject S can write an object O only if the integrity class of the subject dominates that of the object.

Once Alpha can support a mandatory security policy, it is trivial to extend this to include a mandatory integrity policy (no new mechanism is needed; the security lattice need only be suitably extended).

Integrity is also used to mean that the system and user programs and data are protected from corruption, whether accidental or malicious. A variation of the mandatory integrity policy, called *program integrity*, has been proposed for such protection [15]. With program integrity, a subject S can execute an object O only if the integrity class of S dominates that of O. This is because if S executes O, the program O will run with the authorizations and privileges of subject S. But if O is less trustworthy than S, this can lead to either abuse of S's authorizations or corruption of higher integrity data. Program integrity is also easily supported by a mandatory security kernel.

Denial of service [5] refers to the potential ability of a malicious program to consume the resources of the system, thereby preventing urgent or time-critical work from being performed. Alpha's best-effort resource management, while allowing applications to assert their time-varying urgency and importance so that Alpha can maximize the value from its resource allocation, also introduces the potential for a malicious application to cause a denial of service. This can happen because a malicious thread can assert that it has a high importance and urgency and consume the resources of the system, thereby preventing a critical mission from being accomplished. It is desirable that it be possible to trust a thread to accurately assert its time-value function. There are several possible approaches to this problem:

- Use a mandatory integrity policy. With this approach, an integrity class would be assigned to each thread, and Alpha would believe a thread's assertion of urgency and importance according to its integrity class. The integrity class is a measure of a thread's trustworthiness. Thus, how a time constraint is interpreted depends on the integrity of the object that asserts the constraint. Assigning integrity classes to threads means that integrity classes must also be assigned to objects, since the object contains the code that the thread is currently executing. So that a thread cannot be corrupted by reading a low integrity object, the mandatory integrity properties listed above are enforced. In addition, program integrity must be enforced, so that a low integrity object cannot execute with the privileges of a high integrity thread. In Alpha, program integrity can be stated as follows:

> A thread can move from object O_1 to object O_2 only if the integrity class of O_2 dominates that of O_1. A thread can invoke an object only if the integrity class of the thread dominates the integrity class of the object.

This says that a thread cannot move from a high integrity object to a low integrity object and cannot invoke low integrity objects.

- Pre-authorize threads to different maximum levels of importance and urgency. The maximum importance/urgency can be inherited, and a thread cannot spawn sub-threads with greater importance or urgency.

  In keeping with the Clark-Wilson separation of duties [3], the person who assigns the maximum importance and urgency to a thread must be different from the person who wrote the application.

- Use the underlying capability mechanisms to provide a method by which units of resources can be distributed. In this way, the trust needed to assert resource usage is handled in the same way as the trust to access objects. This is a generalization of the "meter" and "spacebank" concepts of KeyKOS [14].

Trade-offs between Timeliness and Security

In the previous section, the issue of trusting a thread to honestly (not maliciously) assert its urgency and importance was discussed, and some approaches were proposed. Once a thread's honesty (trustworthiness) has been established, however, there is an additional problem that it may not be possible to honor its timeliness requests because doing so might violate the mandatory security policy. For example, a high thread may assert a very high urgency and importance, but strict mandatory security would not allow this high thread to be scheduled if it could cause a visible delay to concurrent low threads.

Such a resolution is not always acceptable in real-time systems, however. A trustworthy thread will assert that it is of very high urgency and importance only if it is *critical to the mission of the system* that its urgent timeliness requirements be met. Thus, strictly limiting the potential covert channels in this case may cause an urgent deadline to be missed. One can imagine the severity of such missed deadlines if the urgent process was delivering, say, an intelligence report warning of an enemy unit in the path of an advancing battalion, or was aiming an anti-ballistic missile at an incoming warhead. These hypothetical examples illustrate the important point that the security of the system is a function not only of information flow security but also of how well its timeliness requirements can be met. Thus, it may be that for real-time systems a somewhat different set of criteria by which such systems may be evaluated is needed, as opposed to evaluating such systems strictly according to the *Trusted Computer System Evaluation Criteria* [11].

To resolve these tensions between timeliness and security, the concept of being *important enough to interfere* is postulated (that is, to interfere in a mandatory security sense). Here, if a thread is deemed to be important enough to interfere, then it can be scheduled even if doing so may have effects visible to or detectable by lower-level threads. Whether a thread is important enough to interfere would be assessed by someone knowledgeable about the application, and the resulting designation could be associated with the thread and used by the kernel in its decision making process in much the same manner as are the normal secrecy and integrity attributes. This approach would make Alpha adaptable to these applications-driven trade-offs between timeliness and security.

It is conceivable that the notion of being important enough to interfere could be a dynamic decision made by an intelligent component that would recognize certain changes in the "state" or the environment of the system. An example of such a change might be whether the airborne system is on the ground or in the air. Another example might be the transition from peacetime to war. Different trade-offs between security and timeliness could then be made in the different states.

Even though this intelligent component has to be in the NTCB, and itself introduces the possibility of a new covert channel, the states should change only infrequently, so that the covert channel introduced is much smaller than, say, the potential covert channel introduced by the best-effort scheduling mechanism.

This latter approach is at best speculative, however, in that there are many unresolved issues having to do with putting an expert system in the NTCB [1, 9].

Covert Channels, Denial of Service and Resource Control

It should be obvious from the previous two sections that covert channels, denial of service and resource control are all related. Time decaying (and often, static) covert channels [8] occur as the result of the sharing of resources between threads. Denial of service occurs because of resource starvation caused by threads. It is the precision with which resources can be controlled that determines the ability of a thread to cause a potential information flow and also that determines its ability to cause a denial of service.

For example, consider the issue of concurrency control when accessing elements in a data-base. A standard approach for dealing with the covert channel that would occur when locking data elements is to use multi-versioning, whereby multiple versions of the data elements are generated as they are referenced so that no one version need be locked across multiple access classes. In a real-time system in which applications have precise control over the versions (when they are accessed and their physical memory residency), even this virtualization of data elements results in a covert channel, the size of which may well match that of the channel associated with directly locking the data elements.

Dealing with time decaying covert channels and denial of service has always been a difficult issue. One might think that dealing with these issues for a real-time system must be even more difficult, given that real-time considerations impact virtually all other aspects of system operation. This, however, need not be the case. Non-real-time systems have difficulty dealing with issues of denial of service precisely because they abstract away notions of resources and provide little or no control, externally or internally, to control the usages of those resource so as to prevent denial of service. Since the goal of a real-time system is to precisely manage resources, a real-time system has the potential to handle this class of otherwise largely unsolved problems.

Also, by making explicit a thread's ability to use resources (as a function of true time) the system has the handle on that thread's ability to cause a covert information flow.

It is our hope that a model can be generated that relates the potential information flow resulting from a particular set of resource allocations [10, 16]. In this way, when a particular level of resource control is provided to a thread, the extent of possible information flow can be assessed, and the trust that is needed for that thread can be directly evaluated.

Conclusions

This paper has discussed many issues that arise when multilevel security is applied to real-time systems. The Alpha real-time distributed system was used as a means of illustrating these issues. How Alpha could be made to implement mandatory security policies using its basic capability mechanisms was described. The issues of integrity and denial of service were discussed. An examination of how in real-time systems it may be necessary to make critical trade-offs between timeliness and security was presented. An important point presented in this paper is that the security of a real-time system is a function not only of information flow security but also of how well its timeliness requirements can be met. Thus, it may be that for real-time systems a somewhat different set of criteria by which such systems may be evaluated is needed.

References

[1] T. A. Berson, T. F. Lunt, "Multilevel Security for Knowledge-based Systems," *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, 1987.

[2] K. J. Biba, *Integrity Considerations for Secure Computer Systems.* Technical Report ESD-TR-76-372, USAF Electronic Systems Division, Bedford, Massachusetts, April 1977.

[3] D. D. Clark, D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, 1987.

[4] D. E. Denning, *Cryptography and Data Security.* Addison-Wesley, Reading, Massachusetts, 1982.

[5] V. D. Gligor, "A Note on the Denial-of-Service Problem," *Proceedings of the 1983 IEEE Symposium on Security and Privacy*, 1983.

[6] E. D. Jensen, et al., "Alpha: An Operating System for the Mission-Critical Integration and Operation of Large, Complex Distributed Real-Time Systems," *Proceedings of the 1989 Workshop on Operating Systems for Mission Critical Computing,* ACM Press, 1990.

[7] H. M. Levy, *Capability-Based Computer Systems.* Digital Press, 1984.

[8] K. P. Loepere, "Resolving Covert Channels Within a B2 Class Secure System," *Operating System Review*, July 1985.

[9] T. F. Lunt, T. A. Berson, "Security Considerations for Knowledge-based Systems," *Proceedings of the Third Expert Systems in Government Conference*, 1987.

[10] J. K. Millen, "Covert Channel Capacity," *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, 1987.

[11] National Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria.* DOD 5200.28-STD, Department of Defense, December, 1985.

[12] National Computer Security Center, *National Computer Security Center Trusted Network Interpretation.* Department of Defense, NCSC-TG-005, Version 1, July, 1987.

[13] J. D. Northcutt, *Mechanisms for Reliable, Distributed Real-Time Operating Systems: The Alpha Kernel.* Academic Press, 1987.

[14] S. A. Rajunas, N. Hardy, A. C. Bomberger, W. S. Frantz, C. R. Landau, "Security in Key-KOS," *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, 1986.

[15] L. J. Shirley, R. R. Schell, "Mechanism Sufficiency Validation by Assignment," *Proceedings of the 1981 IEEE Symposium on Security and Privacy*, 1981.

[16] C. Tsai, V. D. Gligor, "A Bandwidth Computation Model for Covert Storage Channels and its Applications," *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, 1988.

TRUSTED XENIX™ INTERPRETATION: PHASE 1

D. Elliott Bell

Trusted Information Systems, Incorporated
3060 Washington Road
Glenwood, Maryland, 21738

Abstract

A set of general results about the mechanisms that provide for need-to-know functionality and policy enforcement in Trusted Xenix™ are presented. These results, centering around a generalization of the discretionary-security-property (called the "weak-discretionary-security-property"), apply to general UNIX®-like systems. [1] An initial mapping of Trusted Xenix TCB calls to model rules is provided.

INTRODUCTION

A full and complete model interpretation for a trusted system (see [TCSEC85]) requires (1) a model representing the security policy to be enforced by the system; (2) a complete list of the calls (or functions or gates) between the Trusted Computing Base (TCB) and the rest of the system; and (3) an interpretation of the TCB-provided calls in terms of the model. An example of an interpretation of this form is [HIS84] In the case of Trusted Xenix (see [GBCC86]), a starting point is [LUCK86]. This paper provides a refinement to that work in (a) basic modeling results and (b) a preliminary interpretation of Trusted Xenix TCB-calls in terms of the rules available in [BLP75] and one additional rule introduced here.

The section MODELING EXTENSIONS introduces the "weak discretionary security property" to allow a faithful representation of Trusted Xenix in modeling terms.[2] The relation between the discretionary-security-property and the weak-discretionary-security-property is then established, and general rule-analysis principles are addressed. One additional rule, an alternate form of rescind-access, is stated and its security-preserving properties are proved. The section INITIAL TRUSTED XENIX INTERPRETATIONS provides an initial model interpretation. A full technical report covering this topic would address system-to-model elements (which system elements are interpreted as subjects, which as objects, and which system elements represent the model's state information). Such a report would also have to support or justify the assertion that the list of TCB-calls was complete and accurate. This paper provides more an overview of such an interpretation: many of the details have been suppressed and a firm conviction about the list's completeness is not asserted. DIRECTIONS FOR FURTHER WORK summarizes the work done and identifies areas needing continuing attention.

MODELING EXTENSIONS

The need for modeling extensions arises from the reflection of mechanisms to support need-to-know policies that is found in [BLP75]. Specifically, the perspective taken on need-to-know mechanisms is parallel to that in the Multics design [ORGA72], although it was not derived from the Multics design. That perspective is that changes to need-to-know permissions will be enforced immediately, even with respect to current accesses previously requested and authorized. In terms of an implementation, this leads to immediate revocation of

[1] XENIX is a trademark of the Microsoft Corporation.
UNIX is a registered trademark of AT&T.

[2] The modeling context is that of [BLP72], [LPB72], [BELL73], [BLP75], and [BELL86].

"current access" within the system on the occasion of a change to the access permission matrix M: any access triple (subject, object, mode) in the current access set **b** would be removed from **b** if a change of state caused mode to no longer be an element of the (subject, object) entry in **M**. This particular feature is present in Multics, but in relatively few other systems. Thus, while the embodiment of a need-to-know mechanism within the model as the state-property called the discretionary-security property (or ds-property) together with the rule $\rho 7$ (rescind-read/execute/write/append) matches Multics faithfully, that combination does not match other systems, such as Trusted Xenix. This section will provide an alternate definition for a need-to-know mechanism that does match the Trusted Xenix functionality (and, in fact, that of any UNIX® system) and derive its properties and implications within the modeling context.

In [MAYE88], the weak-discretionary-security-property was defined in terms of a rule R as follows:

> A rule R, which transitions the system from a current state v = (**b**, M, f) to a new state v* = (**b***, M*, f*), satisfies the weak-ds-property iff $\underline{x} \in M_{ij}$,
>
> whenever $\mathbf{b}^* = \mathbf{b} \cup (S_i, O_j, \underline{x})$ and $(S_i, O_j, \underline{x}) \notin \mathbf{b}$. [p.373]

In this paper, that concept will be recast in terms of actions and extended to appearances and systems in a manner similar to [BLP75].

Definitions: An element of $R \times D \times V \times V$ is called an action.[3] A triple of sequences (x, y, z) that is an element of the system Σ (R, D, W, z_0) is called an appearance.

An action embodies a single change of state, being a relation involving a request to change state, a returned decision token about the request, the new state, and the previous state. The system Σ (R, D, W, z_0) consists of all possible sequences of state-request-decision that satisfy the relation W with respect to actions. An appearance is a single version of events that the system encompasses.

Definition: An action (request, decision, state*, state) satisfies the weak-discretionary-security property (wds-property) provided (subject, object, mode) $\in \mathbf{b}^* - \mathbf{b} \Rightarrow \underline{\text{mode}} \in M_{\text{subject, object}}$.

When an action satisfies the wds-property, every triple added to the current access set **b** was listed as being permitted in the access matrix M at the time of decision. Note that wds-property is not a state property, but is consistent with the spirit of "secure transform", in the sense of McLean [McLE87].

Definitions: An appearance (x, y, z) of the system Σ (R, D, W, z_0) satisfies the wds-property provided every action (x_t, y_t, z_t, z_{t-1}) satisfies the wds-property. The system Σ (R, D, W, z_0) satisfies the wds-property provided its every appearance satisfies the wds-property. A rule ρ is wds-property-preserving provided every action defined by ρ[4] satisfies the wds-property. Call a system Σ (R, D, W, z_0) secure(w) provided it satisfies the ss-property, the *-property, and the wds-property.

The following results are immediate:

- An appearance that is ds-secure[5] satisfies the wds-property.

---

[3] The symbols used here are taken from [BLP75]. R (requests) is the set of inputs to the system; D (decisions) is the set of outputs from the system; V (no mnemonic) is the set of states, each one of which is a triple (**b**, M, f); Σ(R, D, W, z_0) is the system, being the set of all possible changes of state from an initial state z_0 under the constraints of the change-of-state relation W.

[4] The action (request, decision, state*, state) is defined by ρ provided that ρ(request, state) = (decision, state*).

[5] An appearance is "secure" in the sense of [BLP75] provided every state in the state sequence z is secure. The definition of secure in that report was satisfying the requirements for the ss-property, the ds-property, and the *-property. The term "ds-secure" is used to mean that every state satisfies the ds-property: a state v = (**b**, M, f) satisfies the ds-property iff (subject, object, mode) $\in \mathbf{b} \Rightarrow \underline{\text{mode}} \in M_{\text{subject, object}}$.

- A system that satisfies the ds-property satisfies the wds-property.
- A rule that is ds-property-preserving is wds-property-preserving.
- A rule that is security-preserving is security(w)-preserving.
- A secure system is secure(w).

The general results of [BLP75] that deal with the ss-property and the *-property are not affected by the use of the wds-property instead of the ds-property. However, several of the theorems specifically dealing with discretionary-security establishment and preservation have weak-discretionary-security analogues, as below. The results are straightforward and proofs are omitted.

Theorem A3(w): $\Sigma(R, D, W, z_0)$ satisfies the wds-property iff W satisfies the following condition for each action $(R, D, (\mathbf{b}^*, M^*, f^*), (\mathbf{b}, M, f))$ in W:

(i') $(S, O, \underline{x}) \in \mathbf{b}^* - \mathbf{b} \Rightarrow \underline{x} \in M_{s,o}$.

Argument: This result is an immediate consequence of the definition of a system satisfying the wds-property.

Note that theorem A3(w) is simpler than theorem A3, which includes a condition to make sure that newly-non-compliant current accesses are excluded from **b***. Note also that the original condition (i) was phrased in terms of the new matrix M* rather than M; weak-discretionary-security focuses on the conditions at the start of the state transition, rather than on the self-consistency of the resulting state.

Corollary A1(w): $\Sigma(R, D, W, z_0)$ is a secure(w) system iff z_0 satisfies the ss- and *-properties and W satisfies the conditions of theorems A1, A2, and A3(w) for each action.

Theorem A6(w): Suppose ω is a set of wds-property preserving rules. Then $\Sigma(R, D, W, z_0)$ satisfies the wds-property.

Corollary A2(w): Suppose ω is a set of secure(w)-state-preserving rules and z_0 is an initial state which satisfies the ss- and *-properties. Then $\Sigma(R, D, W, z_0)$ is a secure(w) system.

Theorem A9(w): Suppose $(R, D, v^*, v) \in W$, where $v = (\mathbf{b}, M, f)$, $(S, O, \underline{x}) \notin \mathbf{b}$, $\mathbf{b}^* = \mathbf{b} \cup \{(S, O, \underline{x})\}$, and $v^* = (\mathbf{b}^*, M, f)$. Then (R, D, v^*, v) satisfies the wds-property iff $\underline{x} \in M_{s,o}$.

Theorem A10(w): Let ρ be a rule and $\rho(R, v) = (D, v^*)$, where $v = (\mathbf{b}, M, f)$ and $v^* = \mathbf{b}^*, M^*, f^*)$.

(i) If $\mathbf{b}^* \subseteq \mathbf{b}$ and $f^* = f$, then ρ is ss-property-preserving.

(ii) If $\mathbf{b}^* \subseteq \mathbf{b}$ and $f^* = f$, then ρ is *-property-preserving.

(iii) If $\mathbf{b}^* \subseteq \mathbf{b}$, then ρ is wds-property-preserving.

(iv) If $\mathbf{b}^* \subseteq \mathbf{b}$ and $f^* = f$, then ρ is secure(w)-preserving.

Argument:

Given theorem A10, all that needs to be established is that condition (iii) proves that ρ is wds-property-preserving. But the condition $\mathbf{b}^* \subseteq \mathbf{b}$ assures that no new grants of access are made so that the action trivially satisfies the wds-property.

A review of the eleven rules of [BLP75][6] shows that all but $\rho 7$ (rescind access) are trivially wds-preserving so that any set of rules ω from that subset will define a system that satisfies the wds-property. To complete the picture, a rule representing rescinding access in a non-Multics context is needed.

Rule 7(w) (ρ7w): weak-rescind-$\underline{r}/\underline{e}/\underline{w}/\underline{a}$

Request: R = (rescind, subject-1, subject-2, object, $\underline{x}$)

Semantics: **Subject-1** requests that **subject-2**'s access permission to **object** in mode $\underline{x}$ be taken away (where $\underline{x}$ is $\underline{r}$, $\underline{e}$, $\underline{w}$, or $\underline{a}$).

The rule:

$$\rho 7w(R, v) = \begin{cases} (\underline{yes}, (\mathbf{b}, M \setminus [M_{\text{subject-2, object}} \leftarrow M_{\text{subject-2, object}} - \{\underline{x}\}], f)^{7} & \text{if } \textbf{weak-rescind}(\text{subject-1}, v) = \textbf{true}^{8} \\ (\underline{no}, v) & \text{otherwise} \end{cases}$$

Theorem: Rule ρ7w is secure(w)-state-preserving.

Proof: Follows from A10(w) (iv).

Rule ρ7w and the result above provides adequate modeling support for a complete and faithful interpretation of Trusted Xenix TCB calls.

INITIAL TRUSTED XENIX INTERPRETATIONS

The first step in a model interpretation for a system is the identification of subjects, objects, and those portions of the system that correspond to the essential (descriptive) parts of the model being used. In the case of Trusted Xenix, the analogue of the current access set **b**, the access permission matrix M, and the security function f must be identified.

The active entities of Trusted Xenix are the processes. The processes correspond to "subjects" in the model. The passive, data-repository entities in Trusted Xenix to which access is mediated are files, special files, directories, (labeled) pipes, message queues, semaphores, shared memory segments, Xenix semaphores, Xenix shared data segments, access control lists (ACLs), and processes. These entities correspond to "objects" in the model.

The set of current accesses (**b**) are represented in Trusted Xenix by several different data structures. For files, special files, ACL's, named pipes, Xenix semaphores, Xenix shared data segments, and directories, **b** is represented by a set of descriptors in the u_ofile of the per-process u_block. Current access for per-type components are as follows: semaphores is represented by descriptors called semid_ds; message queues, by

[6] (ρ1) get-$\underline{r}$; (ρ2) get-$\underline{a}$; (ρ3) get-$\underline{e}$; (ρ4) get-$\underline{w}$; (ρ5) release-$\underline{r}/\underline{e}/\underline{w}/\underline{a}$; ($\rho$6) give-$\underline{r}/\underline{e}/\underline{w}/\underline{a}$; ($\rho$7) rescind-$\underline{r}/\underline{e}/\underline{w}/\underline{a}$; ($\rho$8) create-object; ($\rho$9) delete-object-group; (ρ10) change-subject-current-security-level; and (ρ11) change-object-security-level.

[7] The notation "A \ B" is from [BLP75] and means "A except as modified by statement B." The notation above indicates that $\underline{x}$ is removed from the matrix entry $M_{\text{subject-2, object}}$, if it was there in state v.

[8] The undefined boolean **weak-rescind** follows the form found in [BELL86] and represents whatever conditions are asserted in the system under consideration to limit the exercise of the rule. An example of the use of **weak-rescind** might be to limit the "invocation" of the weak-rescind rule to the owner of an object. In that case, **weak-rescind**(subject-1, v) = **true** iff subject-1 is the owner of object in state v.

msgid_ds; and memory segments, by shemid_ds. The ipc_perm field of the listed descriptors records the access privileges for different processes.

The access matrix M is stored in Trusted Xenix "by column", using ACL's or more traditional protection specifications. For file-system-like objects, the ACL is identified by the i-node number; the protection specification is contained within the i-node itself. A non-file-system object has its ACL or protection specification associated with its descriptor (semid_ds, msgid_ds, or shemid_ds).

The security function f for subjects (processes) is represented by several values maintained for each subjects, namely the User Maximum Level (UML), the Group Maximum Level (GML), and the Current Process Level (CPL). The subject maximum clearance is the greatest lower bound of the UML and the GML [GBCC86]. The CPL is the current security level. The security levels of objects are recorded in i-nodes (for objects with a file-system representation) or in object descriptors (for those without a file-system representation) [GBCC86].

The identification of those TCB calls appropriate for model interpretation TCB involves the partitioning of all the TCB calls into various categories, some of which are never suitable for interpretation and some of which are. Examples of the first class are (1) TCB calls that return values of internal variables such as ls, ipcs, and df in Trusted Xenix; [9] (2) TCB calls that control the running system, such as c_halt, kill, and fork in Trusted Xenix; (3) TCB calls that deal with practical operations, such as format, the lp subsystem, and star on Trusted Xenix; and (4) TCB calls made by the system itself on behalf of all users, such as syncclock, tsh, and dmesg in Trusted Xenix. A class of TCB calls about which there is debate about their suitability for modeling interpretation are those related to "supporting policies" like audit and identification and authentication (such as auditsh, auditnam, getty, and login in Trusted Xenix). From the latter class of functions that should be dealt with in a modeling interpretation context, there are some that are of a secondary urgency in an evolving modeling interpretation effort, specifically those explicitly reserved to privileged use. Examples in Trusted Xenix are those TCB calls limited to the privileged roles of Security System Administrator, System Operator, AA, and Auditor.

At this point, in the production of a full modeling interpretation cross-reference, the only functions being addressed are those available to unprivileged users and processes. The table below lists the Trusted Processes (TP's) and system calls identified so far in this category, along with the corresponding rule from [BLP75].

TP's

acl	alteration of ACL's	$\rho6$	(give-r/e/w/a) OR
		$\rho7w$	(weak-rescind-r/e/w/a)
c_chmod	alteration of ACL's	$\rho6$	(give-r/e/w/a) OR
		$\rho7w$	(weak-rescind-r/e/w/a)
emkdir	creation of a directory	$\rho8$	(create-object)
mkdir	creation of a directory	$\rho8$	(create-object)
mount	mounting a filesystem	equivalent to a set of	
		$\rho8$	(create-object)
rmdir	deletion of a directory	$\rho9$	(delete-object-group)
umount	unmounting a filesystem	$\rho9$	(delete-object-group)
usrmnt	mounting a filesystem	equivalent to a set of	
		$\rho8$	(create-object)

Kernel Calls

chmod	alteration of permission	$\rho6$	(give-r/e/w/a) OR
	structures	$\rho7w$	(weak-rescind-r/e/w/a)

[9] This sub-class of functions are the so-called "v-funs", or visible-functions, of the field of formal specification. They provide values useful or essential in the use or running of a system, but do not in themselves change the security state of the running system. Such v-funs are, of course, vital in the determination of information flows below the level of abstraction of the model.

close	close a file	ρ5	(release-r/e/w/a)
creat	create a file	ρ8	(create-object)
creatsem	create a semaphore	ρ8	(create-object)
open	open a file	ρ1	(get-r) OR
		ρ3	(get-w)
umount	unmount a filesystem	ρ9	(delete-object-group)

This initial interpretation of Trusted Xenix TCB calls as model rules clearly provides a solid basis for the completion of a full, justifiably complete model interpretation.

DIRECTIONS FOR FURTHER WORK

The work reported in this paper is complete in its theoretical dimension. The definition of the wds-property together with the general results and statement of the new rule ρ7w to account for weak-recision completes the need for theoretical treatment of faithfully representing the security policy enforced by Trusted Xenix. In addition, the interpretation of the model state elements (b, M, and f) is also complete. What still requires attention is the provision of a complete and defensible accurate list of TCB calls cross-referenced to the model rules. The justification of categorizing system calls and internal functions as not requiring modeling interpretation treatment needs to be completed and made rigorous. A last topic not fully resolved is the extent to which the inheritance of current-accesses by a child under a fork-exec process creation necessitates re-evaluating the identification of "process" as being the proper analogue of "subject" in the model.

References

[BLP72] D. Elliott Bell and Leonard J. La Padula, "Secure Computer Systems: Mathematical Foundations," MTR-2547 Vol. I, The MITRE Corporation, Bedford, MA, 1 March 1973. (ESD-TR-73-278-I)

[BLP75] D. Elliott Bell and Leonard J. La Padula, "Secure Computer Systems: Unified Exposition and Multics Interpretation," MTR-2997, The MITRE Corporation, Bedford, MA, July 1975. (ESD-TR-75-306)

[BELL73] D. Elliott Bell, "Secure Computer Systems: A Refinement of the Mathematical Model," MTR-2547 Vol. III, The MITRE Corporation, Bedford, MA, December 1973. (ESD-TR-73-278-III)

[BELL86] D. Elliott Bell, "Secure Computer Systems: A Network Interpretation," *Proc.* 2nd Aerospace Conference, McLean, VA, 2-4 December 1986, 32-39.

[GBCC86] Gligor, V.D., E.L. Burch, C.S. Chandersekaran, R.S. Chapman, L.J. Dotter, M.S. Hecht, W.D. Jiang, A. Johri, G.L. Luckenbaugh, N. Vasudevan, "On the Design and the Implementation of Secure Xenix Workstations," Proc, 1986 Symp. on Security and Privacy, Oakland, CA, April 1986, 102-117.

[LPB72] Leonard J. La Padula and D. Elliott Bell, "Secure Computer Systems: A Mathematical Model," MTR-2547 Vol. II, The MITRE Corporation, Bedford, MA, 31 May 1973. (ESD-TR-73-278-II)

[LUCK86] G.L. Luckenbaugh, V.D. Gligor, L.J. Dotterer, C.S. Chandersekaran, N. Vasudevan, "Interpretation of the Bell-La Padula Model in Secure Xenix," Proc., 9th National Computer Security Conference, Gaithersburg, MD, 15-18 September 1986, 113-125.

[MAYE88] Frank L. Mayer, "An Interpretation of a Refined Bell-La Padula Model for the TMach Kernel," Proc., 4th Aerospace Computer Security Applications Conf, Orlando, FL, 12-16 December 1988, 368-378.

[McLE87] John McLean, "Reasoning About Security Models," *Proc.*, 1987 Symposium on Security and Privacy, Oakland, CA, 27-29 April 1987, 123-131.

[ORGA72] Elliott I. Organick, *The Multics System: An Examination of Its Structure* (The MIT Press: Cambridge, MA, 1972)

[HIS84] "Scomp Interpretation of the Bell-La Padula Model," Honeywell Information Systems, 25 October 1984.

[TCSEC85] *Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, December 1985.

PACL's: An Access Control List Approach to Anti-Viral Security[†]

David R. Wichers[††] Douglas M. Cook Ronald A. Olsson
John Crossley Paul Kerchen Karl N. Levitt Raymond Lo

Division of Computer Science
Department of Electrical Engineering and Computer Science
University of California, Davis
Davis, CA 95616

(916) 752-7004

Abstract—Almost all attempts at anti-viral software have been a reaction to specific viruses that have infected the user community. These solutions attempt to protect against a specific strain or strains of viruses rather than provide general protection against a wide variety of viruses. This paper describes a new, conceptually simple approach that provides a more general solution to the virus problem. Our approach associates with each file in a system an access control list (ACL) that explicitly specifies which programs can modify the file. Thus, a virus cannot modify arbitrary files and its possible effects are greatly reduced. Our approach is unique in the way it uses ACL's to specify which *programs* can access a file; other schemes use ACL's to specify which *users* can access a file and how. We use the acronym *PACL*'s, for Program ACL's, to refer to these ACL's and to our scheme. To see how our ideas can be incorporated into an existing operating system, we have designed an extension to the UNIX[†††] kernel. We also constructed a simulator that has allowed us to gain operational experience with our ideas in a typical user environment. The results indicate that our scheme is a promising approach for preventing the spread of viruses without being too intrusive on users.

[†] This research supported by Lawrence Livermore National Laboratory, the State of California MICRO program, and Deloitte, Haskins, and Sells, Inc.

[††] David Wichers' present address is: Arca Systems, Inc., 2841 Junction Ave, Suite 201 San Jose, CA 95134 (408) 434-6633

[†††] UNIX is a registered trademark of AT&T.

1. Introduction

A computer virus is a program that can 'infect' other programs by modifying them to include a possibly evolved copy of itself [6]. One attribute of viruses that allows them to spread so easily is that a virus inherits all of a user's privileges when the user runs an infected program. Typical operating system protection schemes provide no help in such a case—they protect a user's files from other users, but not from him/herself. Thus, a virus can quickly infect all of a user's files. Even worse, if the user has special system privileges (e.g., 'superuser'), the virus can infect all files on a given system.

A typical virus propagates itself by searching for an uninfected program and copying the viral part of its code into that program so that when the newly infected program is run, the viral code will be executed. To prevent propagation, viruses must be prevented from inserting themselves into other programs. (We assume that the operating system prevents programs, including viruses, from writing directly to disk.)

Two simple observations form the basis of our approach. First, the typical virus carrier is unrelated to the programs that it infects. Second, programs executing on behalf of a user have more privileges than are necessary to complete their assigned task. For example, an infected game program might have the privilege to access all of a user's files. Yet it should only have access to those related to the game, e.g., a score file. Our approach, then, is to restrict a program's privileges to the minimum needed to complete its assigned task. Then, if a program is infected, it will not be able to infect unrelated programs (files).

To impose this *least privilege* restriction, we associate an access control list (ACL) [7, 9] with each file in the system. In our scheme, a file's ACL contains the names of all the programs that may modify the file. We use the acronym *PACL*'s, for Program ACL's, to refer to these ACL's and to our scheme. Thus, to modify (write, append, delete, etc.) the file, a program must be on the file's PACL. Our use of PACL's differs from that found in standard ACL schemes: we store names of *programs*, as opposed to the names of *users*, that can access each file.

The notion of least privilege fits well with common system usage. Users create files using a number of different programs. These files are usually modified only by the programs that create them. For example, consider the typical steps involved in creating, compiling, and linking a C program. To create the program, the user uses his/her favorite editor to create source files. During the entire life of those files, they are only modified by the same editor that created them. When these files are compiled, the compiler generates object files. Each time the program is recompiled, these object files are written over by the same compiler, and not by any other program. Similarly, the linker creates the executable and writes over the executable file each time the program is relinked. This usage suggests that normal files are modified by a small number of programs, usually only one. Of course, more complicated usages exist, but they are less common.

Since the number of programs that need to modify a single file is usually very small, we can keep track of these programs in order to prevent other programs from deliberately or accidentally modifying files. This method is similar to existing computer protection mechanisms based on access control lists. The standard ACL scheme is designed to control how each user's files can be accessed by other users. That is, a file's ACL indicates what users may access the files, and in what ways. If the ACL does not explicitly state that a user is allowed to perform the function requested, then it is not allowed. The difference between this security problem and the virus problem is that a virus security system needs to protect a user from him/herself, not from other users. The virus problem is inherently a problem of integrity, not security. Our *PACL-Integrity* scheme is therefore simpler, associating with each file a list of all programs that can modify the file.

To see how our ideas can be incorporated into an existing operating system, we have designed an extension to the UNIX kernel that incorporates our PACL scheme. We have also constructed a simulator to allow us to gain experience with the PACL-Integrity model without requiring actual changes to the kernel. The experience we have gained shows that the scheme seems reasonable to implement and is not too intrusive on the user.

The remainder of this paper is organized as follows. Section 2 discusses our PACL-Integrity model in more depth. Section 3 describes how the model can be realized in the UNIX kernel. Section 4 presents the simulator and section 5 describes our experience using it. Section 6 discusses the tradeoffs involved in our approach and outlines future work. Section 7 summarizes related work. Finally, section 8 contains some concluding remarks.

2. The PACL-Integrity Model

The PACL-Integrity model associates a PACL with each file on the system. The PACL for a given file names all programs that have the privilege to modify the file. When a file is created, its PACL is set to contain the name of the program that created the file. During the life of the file, the file's PACL can be changed only by a trusted utility program. This utility allows a user to tailor the protection mechanism to meet his/her needs.

The success of a protection scheme depends on how intrusive users find it. A scheme that is too intrusive will effectively render a system unusable. For example, requiring an explicit acknowledgement from a user each time any file is to be accessed might be a secure scheme, but it is not usable. Moreover, if a scheme that is too intrusive provides a means by which the user can disable it, then users will simply run with security checks disabled, effectively rendering a system insecure.

To make our approach secure yet usable, we include a number of 'user-friendly' features. These features simplify common usages of the PACL-Integrity mechanism. The first feature is an inheritance mechanism that allows a user to define a default PACL for a directory. Any file (or subdirectory) created in this directory inherits the directory's default PACL, as well as the name of the program that created the file. This feature allows the user to tailor a directory to the type of work being done in it. An entire system (or account) can be tailored in this manner by creating a default at the root (or home) directory and then building directories below it.

The second feature allows a user to specify a global inheritance policy. The user can define a default PACL for any file based on its extension (suffix). For example, a UNIX object file is typically created by an assembler or compiler and given the extension '.o'. Later, the linker reads in a number of object files, links them together, and generates executable code. When it has successfully generated an executable, it sometimes will remove the object files as they are no longer needed. Since the object files were created by the compiler, their PACL's will contain the name of the compiler, but not that of 'ld' (the linker). With the extension-based default mechanism, the user can define a default for '.o' files that contains 'ld', thereby allowing the linker to remove unwanted object files after it has created the executable.

The third feature allows the user to enable/disable the PACL mechanism for a particular file. This feature is provided by associating a flag with each file. If this flag is enabled, the normal PACL security rules will be applied to that file. If the flag is disabled, then all PACL security rules for the file are ignored and only the 'normal' security rules will be used when the file is accessed; i.e., any program with appropriate access rights can modify the file.

The final feature allows a user to temporarily disable the PACL mechanism for all of his/her files. It also allows the system administrator to temporarily disable the PACL mechanism for the entire system. This feature is needed to facilitate programs that need to modify many or all of a user's or system's files. For example, a utility program that restores files from backup tapes will typically modify many files during its execution.

These features are provided to allow the system to be tailored to meet each individual user's needs. Once defaults have been set up correctly, each user should be able to use the system while being protected from viruses, without being unduly inconvenienced by the PACL mechanism.

The PACL-Integrity mechanism makes several basic assumptions about the underlying hardware and operating system. The devices on which programs are stored (e.g., disk) must be protected so that they can only be accessed by kernel code. Without such protection, a virus could write directly to a device, bypassing all protection mechanisms. This requirement rules out the possibility that this type of system would be viable in some personal computer environments where direct disk access is not protected, for example. The operating system itself must check all file accesses to make sure the PACL security rules are enforced. It must also protect the PACL's themselves from illegal modification. The hardware and operating system must also protect against standard attacks, such as modifying system buffers or kernel code.

3. A PACL-Integrity Model for UNIX

3.1. Overview

Our PACL-Integrity model can be implemented for UNIX by extending the kernel. The PACL scheme must be included in the kernel to ensure that all file accesses are checked. The current UNIX protection mechanisms, based on user names, are still enforced. If an attempt to write satisfies the existing security rules, the PACL mechanism then further verifies the validity of the access.

When a file is created, its PACL is created as well. A file's PACL is stored as part of the header information (i.e., *inode*) of the file, just like the mode bits, owner, size, date, and time fields. Since the PACL is part of a file's inode, the PACL information for a file is removed when the file is deleted, which simplifies the task of PACL maintenance.

The kernel builds the PACL for a new file from three items. The first item put in the PACL is the name of program that creates the file. In UNIX, a program's name is its complete pathname. For example, the editor program 'vi' in the directory '/usr/ucb' has the name '/usr/ucb/vi'. The second item put in the PACL is the default PACL of the directory in which the file is created. The final item put in the PACL is the default, if any, for the new file's extension. (Note that the defaults put in the PACL are those in effect when the file is created; if the defaults are later changed, the PACL's of existing files are not modified automatically.)

The specific kinds of access for which the kernel must check include opening a file for writing and unlinking a file. The former gives the program the privilege to modify the file in any manner while the latter deletes the file. We consider deletion a form of modification.

3.2. New System Calls

Nine new system calls give programs the ability to interact with the PACL mechanism. The first system call, *setppriv()*, is a privileged call that sets the state of the current process into a mode that allows it to call the other new system calls. (This method is analogous to a process setting its user-id to root in regular UNIX.) Without executing this initial call, a process is not allowed to use any of the other system calls that interact with the PACL's, with one exception described below; in such a case, they simply return an error to the calling process. The only programs that are allowed to use *setppriv()* are the programs listed in the file '/etc/paclprivs'. One example of an entry in this file is the utility program described later.

The second call, *paclenable()*, is used to enable or disable (based on its argument) the entire PACL mechanism for the given process and its children. If the initial system process (*init*) disables the PACL mechanism, then the effect is that the PACL mechanism is disabled for the entire system since all processes are children of *init*.

The third call, *clrppriv()*, removes a process from PACL privileged mode. It allows the process to relinquish its privilege when no longer needed. The two calls *setppriv()* and *clrppriv()* allow programs to create critical regions in their code where they have privilege to access PACL's. Outside of these regions, PACL privileges are not necessary and hence should not be enabled.

The fourth call, *getppriv()*, is the only call that will not return an error if *setppriv()* has not been previously called. It tells the currently running process whether or not it is currently in PACL privileged mode, i.e., the process successfully called *setppriv()* without calling a corresponding *clrppriv()*.

The next two new system calls allow a program to manipulate PACL's. Only the owner of a file can change its PACL. The first, *addpacl()*, adds a program name to a given file's PACL. The second, *delpacl()*, deletes a program name from a given file's PACL. These system calls also allow a file's owner to enable/disable the PACL mechanism for a particular file.

The remaining three system calls allow a program to query a file's PACL in various ways. These calls can only be executed by the file's owner. The first, *getpacl()*, returns a list of all the program names in a file's PACL. The second, *verpaclm()*, determines if a specified program has the privilege to modify a given file. It compares the program name with those in the file's PACL, handling links if the filename provided is a link to another file. The third, *verpaclr()*, determines if the specified program has the privilege to remove a given file. It is similar to *verpaclm()* except it does not traverse links because any remove reference to a link would be removing the link, and not the file to which the link points.

3.3. The *ch* Utility

The above eight system calls provide the means for a system program to manage PACL's. The utility program, *ch*, described below uses these calls and is an example of a type of user interface that can be provided for user interaction with this mechanism. *ch* is listed in '/etc/paclprivs' so that it is authorized to use these PACL system calls on the user's behalf.

To use the *ch* utility, the user must first enter his/her password. We make the assumption that a virus can assume a user's login name but it does not know the user's password. Otherwise, we cannot distinguish a virus from a legitimate user.

ch allows the user to:

- add/remove program names from PACL's;
- display the contents of PACL's;
- set/clear the enable flag in PACL's;
- modify the default PACL's for directories and file extensions; and
- temporarily turn off the entire PACL mechanism (e.g., for that user during a single login session).

These features correspond to those described in section 2. Several additional features make the utility more usable. One feature allows the user to traverse the directory structure; a user can, therefore, move to different directories without exiting the utility. A second feature is that *ch* provides all the remove privileges that exist in a normal shell. The 'rm' (remove) program may not have privilege to remove most files; i.e., it may not be in the PACL for every file. The utility, therefore, provides an 'rm' command with functionality equivalent to that of the 'rm' program. Without such a command, the user would need to add the 'rm' program to a file's PACL, exit the utility, and then use the 'rm' program to remove the program. For the same reason, the utility also provides an 'rmdir' (remove directory) command. Basically, *ch* provides a subset of the normal shell commands along with the features described above that allow the user to tailor the PACL security system. If the user executes a program from within *ch*, a new process is created to execute that program. This process is subject to the rules that apply to the new program, not those that apply to the *ch* program. Other utility programs can easily be generated by the system administrator by writing programs using these system calls and then adding the program names to '/etc/paclprivs'.

3.4. The Role of the Superuser

In existing UNIX systems, the superuser—e.g., the 'root' account—may bypass the normal protection mechanisms. Having root privilege is not sufficient to override the PACL protection mechanism in our system. In particular, a user (or would-be virus) executing as root can only disable the PACL mechanism using the *ch* utility, for which it must give the root password. A program running as root must, therefore, be listed in a file's PACL in order for that program to have the privilege to modify that particular file.

This approach limits the damage potential of a virus that somehow acquires root privilege.

This restriction, however, requires us to change the current method by which the superuser changes the root password. Currently, the superuser uses the 'passwd' program to change the root password. The password program prompts for the new password without asking for the old one. Thus, any user (or program) that acquires root privilege can change the root password without knowing the previous password. Such a user could then use *ch* to break system security. Therefore, we now require *passwd* to ask the superuser for the old root password before changing it. This additional requirement prevents a virus from changing the root password without knowing the previous password. The one exception is that the superuser can change the root password without entering the old password when the system is brought up in console (single user) mode. This exception exists to allow access to a system in case its password file gets corrupted.

Since the success of our anti-viral scheme depends heavily upon password security, viruses must be prevented from obtaining passwords. In our scheme, the password file itself is protected so the only programs allowed to modify it are 'passwd' and those that modify information about users (e.g., user names, phone numbers, etc.). A new user can be added according to one of two methods. The first method is to add an editor, say 'vi', to the password file's PACL, then edit the file to include the new user, and then remove 'vi' from the PACL. This method is not a major inconvenience to the system administrator if new users are added infrequently. On the other hand, the above method is cumbersome for a system where new users are added frequently. A better method, then, is to write a new utility program that is specifically designed to add users to the password file and to place the name of this new utility in the password file's PACL. Execution of this utility program should be restricted to only the system administrator.

4. A PACL-Integrity Model Simulator

We constructed a UNIX-based simulator to allow us to experiment with our ideas. Building a simulator required less effort than making kernel modifications would have. Doing so also had no impact on other users of the system as making kernel modifications would have.

The simulator consists of modifications to the standard C library. It is not a program itself. The simulator library contains modified versions of the normal system calls that deal with files (e.g., open) and code for the new system calls dealing specifically with PACL's. The normal system calls take the same arguments as usual. Thus, the simulation environment is transparent to most programs; they just need to be linked with the new library. The code in the library routine that handles a normal system call is an interface to the original routine that handles the system call. It first does whatever PACL checking is needed and then calls the original routine, which has been renamed.

The simulator maintains a *virtual root*. The virtual root allows any directory to act as the root directory during simulation experiments. The simulator maps any reference to root (i.e., a pathname that starts with '/') to the virtual root. For example, if '~cook/test/pacl' is the virtual root, the simulator maps '/bin/cp', the copy program, to '~cook/test/pacl/bin/cp'. Using a virtual root lets us test the PACL mechanism by defining a subdirectory that contains an entire UNIX environment, i.e., all the standard system programs. The programs in such a subdirectory are linked with the simulation library. Using a virtual root also allows a user to experiment on a system without requiring root privileges. Further, it allows several users to run experiments at the same time as each one can define their own virtual root. (The idea of a virtual root is similar to the UNIX 'chroot' command except it works on a per-process basis and does not require root permission.)

Section 3 described how a file's PACL information is stored as part of its inode. That is not possible without kernel modifications. The simulator, therefore, stores the PACL information for file *x* in another file, *x.pacl*. Similarly, the default directory PACL for a directory is stored in the file 'default.pacl'. These files are not visible to the user when running under the simulator. They can only be created by the simulator for its purposes.

The default PACL information for file extensions is stored in an environment variable. The simulator uses this information along with the default directory PACL information and the executing program's name (see below) when creating the PACL for a new file.

The simulator needs the name of the currently executing program for creating PACL's and comparing access rights. The simulator maintains that name in an environment variable. The variable is set in the simulator library's *execve()* system call, which is invoked whenever a process is created. It is examined whenever a process executes a system call that needs PACL privilege. This method is insecure because a process can modify its environment variables—e.g., a process can change the simulator's idea of its name to gain illegal access to a file. However, the method is adequate for our simulation purposes. In a kernel implementation, the name of the currently executing process would be stored so that only the kernel could modify it.

5. Experience

The additions to the C library to form the simulator library required about 1000 lines of code. The additional code intercepted system calls, translated pathnames to virtual root-based pathnames, and checked PACL permissions.

We have used the simulator in a number of situations and have gained some idea as to the effectiveness and intrusiveness of our PACL scheme. Our tests fall into two categories: general interactive use and installation of software systems. In the first kind of tests, users performed the activities they normally would on a system—i.e., developing programs, writing papers, etc.—and would also occasionally attempt to defeat the PACL mechanism. In these tests, the PACL mechanism worked as it was intended: It was successful in preventing simulated viral attacks without being too intrusive. One observed drawback, however, was that users needed to be aware of the PACL mechanism. One common problem, for example, was that users had problems removing files since the remove program 'rm' was not on the PACL of the file being removed. The utility program proved useful, but it requires the user to learn a new tool.

The second kind of simulator test—software installation—was also generally successful. We attempted to install two large software systems, GNU Emacs [11] and the SR concurrent programming language [1], in the simulated environment. Although the installations uncovered several problems with our simulator, they did demonstrate the validity of the overall design of our PACL scheme.

6. Discussion

Our PACL-Integrity model is an integrity model only. As such, it protects files from illegal modification but not from exposure. One advantage of it being just an integrity model is that the system is greatly simplified. PACL checks occur when the file is opened for writing and the checks themselves are very fast. A PACL check consists of looking up the program name to see if it exists in the file's PACL. Since a file's PACL will typically be very short, that check will be very fast and the space overhead involved per file will be minimal. (For simplicity, we have restricted in our initial designs a fixed sized space to store the PACL for each file.)

Our PACL scheme is obviously not perfect. It can be defeated by exploiting existing operating system security loopholes or trojan horses. It also requires that the PACL's for the system are set up correctly, which requires user and system administrator cooperation.

One potential vulnerability of our PACL scheme is that a virus could invoke other, more trusted programs to do its dirty work. For example, a virus could send commands to infect files to 'vi'. One possible solution to this problem would be for programs to impose restrictions on how they operate; e.g., 'vi' might accept only interactive input rather than accepting input from another program. A more general solution, however, will require further study. Even with this vulnerability, our PACL scheme substantially reduces the vulnerability of the overall system.

One issue that we have not fully resolved is exactly what constitutes the name of a program. As described earlier, the name of the file is its complete pathname. However, a single file on disk can have many different names (i.e., paths to it) through *hard* or *symbolic* links. A hard link establishes another name for a file by having another directory entry point to the file's inode. A symbolic link is a file whose contents is a file name; when the symbolic link is opened, the kernel instead opens the contents of the link. Given such possibilities of multiple names for a given file, which name or names should be used in the PACL checks needs further study.

Another issue related to naming is what to do when the name of a program changes. Since the PACL's store full pathnames of a file, they must be changed. One possible solution is to extend the *ch* utility to provide a rename option. However, that is expensive as it needs to search all PACL's for all files in the system. Moreover, it requires user intervention. Another possible solution is to store in the PACL the program's inode number instead of its name. A sequence number would also need to be maintained for each inode to distinguish between different uses of an inode, e.g., to ensure that when a free inode is reused, it does not accidentally allow access to the wrong files. Renaming is important because it occurs fairly frequently, although more often for user programs than for system programs. Another related issue is what to do when a new version of a program is installed. If program names are stored, then the PACL scheme works fine. If inode numbers are stored, then they would need to be updated, which is expensive as described above. One final related issue is how to handle deletion of programs. When a program is deleted, it should be removed from all PACL's in which it appears. Otherwise, a virus might install a new program in that place. On the other hand, if a program is deleted just before a new version of it is installed, then the cost of cleaning up all PACL's should be avoided. These issues are important and related to one another. Further work and experience is needed before the 'right' solution can be determined.

One possible objection to the entire PACL approach is that it requires future knowledge to be totally effective: it must know *for all time* what programs will need to access what files. That is clearly impossible, especially since new programs can be added to a system. For example, suppose an existing file system has its PACL lists set up so that the C compiler, say located in '/bin/cc', is allowed to create '.o' files. If an alternate C compiler such as GNU's, say located in '/usr/local/gcc', is added to the system, then the PACL's for all '.o' files should be updated to also allow the new compiler to modify those files. Requiring users to perform such modifications of PACL's is not attractive; a tool to automate such modifications should be straightforward to develop.

The use of the extension-based defaults and the directory inheritance in our PACL scheme provides a flexible enough environment for a user to perform most tasks without considering the PACL's. A more complicated task, especially one involving files with nonstandard extensions, may require the user to modify PACL defaults. However, once that is done, the task can be completed with little difficulty. User intervention is typically only required to set up defaults for a new task; repetitions of that task do not require further intervention.

The initial setup of a system that uses PACL's is also a nontrivial task. In particular, the system administrator must determine PACL's for each system file, and directory and extension defaults. Fortunately, such work needs to be done just once. Of course, this problem will go away if PACL systems become the standard; vendors would then ship PACL-equipped systems already set up.

7. Related Work

Recently, Eugene Bacic [2] proposed a similar scheme that was developed independently from that proposed in this paper. His mechanism also associates an additional ACL-like list with each data object to provide integrity controls by constraining the programs that can manipulate an object. His paper addresses the subject from a more theoretical slant than the application specific (UNIX) approach discussed here.

Karger [8] and Boebert and Ferguson [4] have proposed solutions similar to each other that attempt to address the Trojan Horse problem. Both solutions interpose a protected subsystem between programs and the filesystem to protect the filesystem. They are related to the mechanism we described in that they use some type of knowledge base (in our case the PACL's) to make access control decisions based on the user executing the program, the program being executed and the files being accessed. The methods they proposed to generate and use this knowledge base are quite different. The intent of our solution is to generate this knowledge base as simply and easily as possible, while making it simple to design, simple to build, simple to maintain, powerful to use and simple to understand.

In a landmark paper, Clark and Wilson [5] introduced a model of integrity that is based on control of which actions users can perform on particular data items. The model is applied to constrained data items (CDI's) and only allows access to these CDI's through Transformation Procedures (TP's). The central system enforced property required by Clark-Wilson is that the system maintain a list with entries which describe for a user-id, TP pair, which CDI's the user can access with the given TP. The system must further ensure that no CDI can be manipulated except through a TP. The PACL mechanism described in this paper can directly support the enforcement of Clark Wilson controls with the restriction that there is only a single list of programs allowed to access a given file rather than a separate list for each user or group of users. Using the normal access control mechanism, the users which can access a CDI are described by the standard permissions, and the TP's which can access the CDI are listed in its PACL.

8. Conclusions

The PACL scheme presented in this paper is a step toward providing protection against viruses. It is an attractive approach since it is relatively simple, both conceptually and to implement, and it aims to protect against all viruses, not just specific strains. The simulator allowed us to gain experience using our scheme. This experience has been quite positive and shows that our approach is feasible. Although this paper describes our scheme and experience in the UNIX environment, our PACL scheme also applies to other operating systems. We plan to gain additional experience using the simulator, and then to implement our scheme in the kernel. Our other plans include extending the PACL scheme to a networked environment and considering how a collection of systems—some using our PACL scheme and some not—will interact. At a broader level, we are also investigating ways of combining various protection schemes (e.g., ACL's, capability lists, type enforcement schemes [10], integrity labels [3, 10], and POSET model [6]) into one unified scheme. The unified scheme will allow flexibility in choosing which scheme is appropriate for a given problem.

Acknowledgements

Bill Wilson, Seth Abrahams, Doug Mansur, and John Wagner provided very useful input on our ideas.

References

[1] G.R. Andrews, R.A. Olsson, M. Coffin, I. Elshoff, K. Nilsen, T. Purdin, and G. Townsend. An overview of the SR language and implementation. *ACM Trans. on Prog. Languages and Systems*, **10**(1), pp. 51-86, January 1988.

[2] E.M. Bacic. Process execution controls as a mechanism to ensure consistency, *Proceedings of the 5th Computer Security Applications Conference*, Tucson, AZ, Dec. 1989.

[3] K.J. Biba. Integrity considerations for secure computer systems. MTR-3153, Mitre Corporation, Bedford MA, 1975.

[4] W.E. Boebert and C.T. Ferguson. A partial solution to the discretionary trojan horse problem, *Proceedings of the 8th National Computer Security Conference, National Bureau of Standards*, Gaithersburg, MD, Oct. 1985,

[5] D.D. Clark and D.R. Wilson. A comparison of commercial and military computer security policies, *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, Oakland, CA, April 1987.

[6] F. Cohen, Computer viruses—theory and experiments. *Proceedings of the 7th DoD/NBS Computer Security Conference*, National Bureau of Standards, Gaithersburg, MD, USA, September 1985, pages 240-263.

[7] D. Denning. *Cryptography and data security*. Addison Wesley, Inc., Reading, MA, USA, 1982.

[8] P.A. Karger. Limiting the damage potential of discretionary trojan horses, *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, Oakland, CA, April 1987.

[9] E.I. Organick, *The Multics System: An Examination of Its Structure*. MIT Press, Cambridge MA, USA, 1972.

[10] M.A. Schaffer and G. Walsh. LOCK/ix: on implementing UNIX on the LOCK TCB. *Proceedings of the 11th National Computer Security Conference*, 1988.

[11] R.M. Stallman. *GNU Emacs Manual, Sixth Edition, Version 18*. Free Software Foundation, Cambridge, MA, March 1987.

STATIC ANALYSIS VIRUS DETECTION TOOLS FOR UNIX SYSTEMS [1]

Paul Kerchen, Raymond Lo, John Crossley, Grigory Elkinbard, [2]
Karl Levitt, Ronald Olsson
Division of Computer Science
University of California
Davis, CA 95616

Abstract

This paper proposes two heuristic tools for detecting viruses in a UNIX environment. The tools would be used to detect infected programs prior to their installation. The tools use static analysis and verification techniques. One tool, the *detector*, searches for duplication of operating system calls. A program compiled and linked from source code (such as C) makes calls to standard library routines for operating system services; relevant to detecting viruses are calls on files services, such as open and write. Such object code will contain only one instance of the standard library subroutine for each type of service requested by the program. A virus would most likely carry along its own system calls; hence an infected program would have duplicate calls to the file service and is easily caught by the detector. The second tool, the *filter*, uses static analysis to determine all of the files which a program is capable of writing to. By knowing what files a program can and cannot write, one can decide whether or not that program is suspicious. The paper discusses the features and shortcomings of both tools and gives some implementation details related to the detection of UNIX viruses. In order to defeat these tools, a virus would have to be quite complex and, if successful in avoiding detection by these tools, accept limited propagation. The tools are also useful for detecting more general malicious code, such as Trojan Horses.

1 Introduction

Ideally, one would like to be able to detect an infected program without having to execute it and without noticeably impairing the performance of the system. Some virus detection techniques (see [6] and [7]) rely on run-time checking of program behavior, but employ auxiliary hardware to avoid a performance penalty; the hardware can be viewed as a generalization of the familiar watchdog timer. However, these run-time methods potentially expose the system to a virus which is able to do its damage before being detected. Other run-time techniques (see [2], [3], and [8]) do not allow a program to execute if it fails to pass certain tests; these methods are useful, but they may introduce an unacceptable amount of overhead to the execution time of programs. Typically, these methods involve protecting programs stored on a disk with cryptographic checksums. Another method [10, 11] queries the users at runtime for all file modifications or requires users to identify the programs that can write to his files. Most virus detection techniques have serious limitations because they detect and inhibit the spread of viruses, not their presence. They cannot be applied to programs which are obtained from unreliable sources since they all rely upon having a *clean* copy of the program available for comparison, or they require user interaction at runtime, or they require access protection mechanism absent from most operating systems. Other approaches (e.g.

[1]Supported by grants from Lawrence Livermore National Laboratory, the State of California MICRO program, and Deloitte Touche, Inc.

[2]Currently at Amdahl Corporation, Sunnyvale,CA

virus scanners) cope only with known viruses or virus strains. Our approach attempts to identify viruses through detecting their discerning characteristic in an infected program.

Our approach involves the analysis of a program prior to installation, the analysis attempting to identify suspicious code. By statically analyzing a program, one can in principle determine whether a program contains suspicious code, regardless of whether or not *clean* code is available. This paper presents two static-analysis methods under implementation for detecting suspicious code indicative of a virus. These methods are based on the following premises:

- Source programs are linked with the standard library during compilation. In most systems, the operating system services, e.g. file open, file read, file write, are provided to the user in the form of library functions. Hence a compiled and linked program should contain at most one instance of the trap instruction to the operating system for each system call. Simple viruses, attaching themselves to the beginning or end of a program, would carry along their own trap instructions. Infected programs would have duplication of such trap instructions for some system calls.
- A program containing a virus will contain calls to write the virus to storage, e.g. to the disk, operating system memory, or to uninfected files. Suspicious code, then, could cause the program to write to files the program under investigation is not expected to write to. By enumerating all of the files a program can potentially open, the user of the program is alerted to potentially suspicious code before he runs the program.

These two points form the basis of the two UNIX tools being presented here. The *detector* tool examines a program to determine if it contains any duplicate instances of operating system services (such as file operations like read and write), while the *filter* tool will examine a program to ascertain which files the program can write. These tools are promising because they can detect a large class of viruses and limit the propagation of others. Although these tools are limited by a number of factors, they form a firm foundation upon which more sophisticated tools may be built.

To date, the detector tool has been implemented and tested on several programs with promising results; we have determined that all but one of the UNIX utilities on our Sun-3 workstation running SunOS 3.4 have no duplicate trap instructions. Furthermore, the detector has detected a handcrafted virus that is typical of UNIX viruses. A prototype of the filter is under development, but it has been hand-simulated on several utility programs.

The remainder of this paper discusses the basic approach of the detector and the filter tool. The discussion includes the assumptions attendant to each tool as well as the implications of these assumptions. The implementation of the detector is discussed, giving details about problems and results of experiments performed with it. A discussion of a simple UNIX virus is also given to facilitate the understanding of the implementation. Next, the concepts behind the filter are explained in detail. The shortcomings of each tool are discussed and extensions of the tools are suggested as work for the future.

2 The Detector

2.1 Basic Approach

The purpose of the detector is to identify duplicate calls to operating system services; duplicated calls might be in an executable program and be indicative of a virus that has linked itself to the program. The first step in the detector's analysis is to disassemble a program into its equivalent assembly language representation. The next step consists of finding all instances of code which perform some operating system service. If two different pieces of code are found to contain the same operating system services, then this condition is flagged as a duplication of services. For most programs, it is reasonable (and necessary) to make the following assumptions:

1. The program uses a standard interface for communicating with the operating system.
2. The program is generated with a compiler.
3. The source program does not call the operating system directly through a trap, instead it uses the operating system interface in the standard library.
4. Virus code can only occur in the code (text) segment of a program.

Assumption one ensures that determination of duplication of services will be relative straightforward. If all programs use the same format for using system services, the detector can always determine what the service is. For instance, in most implementations of UNIX, system calls are performed by pushing the system call number onto the stack and then executing a trap to the operating system. If the system call number is always pushed immediately before the trap is executed, the detector simply has to examine the instruction preceding the trap to determine which service is being used. If a program does not follow such a scheme but instead handles each system call in a different way, the detector must then symbolically execute the program to determine the contents of the stack at the time of the trap instruction—a more difficult and potentially intractable problem. Fortunately, most versions of UNIX use a standard calling scheme. Thus, this assumption is only restrictive for those programs which do not use the standard calling scheme, such as some programs written in assembly language.

The second and third assumption are necessary to ensure that a legitimate, uninfected program will not have any duplication of services. Executable programs linked with the standard library will have one routine which handles all requests for a given operating system service. Any time the program needs a service, it effects the appropriate preparations, such as pushing the other information required for the call (e.g. arguments) onto the stack, and then calls the routine which performs the service. This technique to handle system service call is very common and not confined to UNIX.

For portability and upgrade compatibility reasons, a compiler does not generate code that interface with the operating system directly. Instead, the compiler will treat a system service call as a subroutine provided by the standard library. The actual operating system interface code, i.e. the system trap, resides in the library subroutine. Therefore,the actual interface should appear at most once for each system call in any compiled program. [3]

Finally, assumption four stems from a consideration of file formats and their related restrictions under UNIX. Typically, UNIX uses three file formats for executable files: OMAGIC, ZMAGIC, and NMAGIC. The first, OMAGIC, is obsolete and rarely used. In this format, the text segment is non-sharable and not write protected, so the data segment is immediately contiguous with the text segment. The second, ZMAGIC, is the default format produced by *ld*, the link editor. For this format, the text and data sizes must both be multiples of the page size since the pages of the file are brought into the running image as needed. The third format is similar to the second except the data and text segments are not required to be multiples of the page size; the entire image is preloaded into memory at run time. Most versions of UNIX enforce segmentation of code and data, meaning that executable code and non-executable data must reside strictly in their respective segments. Furthermore, the text segment is not writable during run time and execution of the data segment is not allowed. As a result of these restrictions, a virus which infects a program must do so by placing all of its code into the text segment; it cannot hide any code in other parts of

[3] In order to defeat the detector, a virus would have to use the operating system calls of the program it is attempting to infect, rather than trivially attaching itself to the beginning or end of the program. Later, we discuss ways to catch attempts to defeat the detector.

the file. NMAGIC and OMAGIC format files, therefore, are somewhat more resistant to viruses than ZMAGIC format files since no unused space is available for a virus. However, a virus may still be able to infect such files if it can somehow hide the increase in the size of the host program (perhaps through a flaw in the operating system or by compressing the original code to obtain space). ZMAGIC format files are even more vulnerable. For instance, under SunOS 4.0 the page size is eight kilobytes, meaning the average ZMAGIC format program will have approximately four kilobytes of zero-padded space in both its text and data sections. This space is large enough to hold a fairly complex virus written in assembly language. However, in all three cases, the virus code must still appear in the text segment, making it detectable by the detector. If all of these conditions are met, then the detector can be used to determine if the program under consideration contains any duplication of system calls.

2.2 Implementation and Results

A prototype of the detector has been implemented on a Sun 3 workstation running SunOS 3.4 and has been tested on several of the standard programs from */bin*, */usr/bin*, and */usr/ucb*, but its application is not limited to UNIX systems. This prototype, called Snitch, is written in the C and Icon programming languages and consists of two major modules: the disassembler and the analyzer. The first module, the disassembler, takes an executable program as input and produces the equivalent Motorola 68020 assembly language representation as output. The second module, the analyzer, takes the output from the disassembler and examines it for duplicated code.

For SunOS 3.4, a system call is performed by pushing the system call number onto the stack and then executing a trap instruction. Because the call expects the top of the stack to contain the number of the call to be made, determination of duplication of services becomes straightforward: one only needs to backtrack from the point of the trap to determine the last item pushed on the stack; that item will be the system call number. Furthermore, most of the standard library routines push the system call number immediately before executing the trap, making the analysis phase even simpler. The analyzer reports any duplications found as well as the number of occurrences of all system calls.

The results of the experiments performed on Snitch are as follows. Approximately one hundred programs (mostly UNIX utilities) were tested for duplication of services with some of them infected with a simple virus (described in Section 3.2). All of the infected programs were found to have duplicated system calls, while only one uninfected program was flagged as having duplication of services: */bin/csh* contained two instances each of the *getgid* and *getuid* system calls. One may conjecture that such duplication occurred because of post-linking binary patching. Since the duplicated services were not of a serious nature; for a program as large as the C-shell, such an occurrence should not be surprising or indicative of malicious code.

2.3 A Simple Virus

For purposes of testing Snitch, a simple virus was created which infects SunOS 3.4 executables. The virus is considered simple because it makes no effort to conceal itself and it does not use a sophisticated method for replication and propagation, although it is capable of avoiding multiple infections of the same program. Basically, the virus works as follows: First, the virus determines whether it has previously infected the target program. Under SunOS, executables have a standard header which contains format information, start-up code, a branch to the user's code, and then clean-up code. The format information tells in which format (OMAGIC, ZMAGIC, or NMAGIC) the file is arranged. The start-up code initializes environment variables and other constructs while the clean-up code restores the old environment and makes a smooth return to the shell. All of this information is common to most executables and of a constant length. Therefore, the branch to the main body of code always occurs at a certain offset from the beginning of the text segment.

Furthermore, the user code always immediately follows the clean-up code, making the branch address the same for all programs. Thus, to determine previous infection, the virus simply examines the location in the text segment where the branch instruction occurs (bytes 70-73) and determines if the address is the standard address (20A0 hexadecimal). If it is, the virus commences the infection process.

Next, the virus determines if it has enough space to infect the program without overwriting any legitimate code or increasing the size of the program. The only format which allows any zero-padded space is the ZMAGIC format; if the file is not in ZMAGIC format the virus exits and passes control to the legitimate code. If the file is in ZMAGIC format, the virus determines whether there is zero-padded space at the end of the text segment. This task is accomplished by looking for zero-padded space of length N between the end of program and the end of the text segment, which is multiple of 8K bytes. N is the length of the virus code.

Finally, assuming there is enough room, the virus copies itself from the host program into the target program by copying the last N bytes from the host program's text segment. It then changes the branch instruction in the start-up code so that the virus code is executed after the start-up code and before the legitimate code. Five system calls are used by this virus (open, lseek, read, write, and close) and its length is approximately 150 bytes. A program infected with this virus is easily detected by the detector.

2.4 Limitations of the Detector

The most obvious way of defeating the detector is simply to make the infected program not have any duplication of actual interface to the operating system; if the virus uses the existing services it cannot be detected with the detector. Use of existing services would be simplified if the symbol table information was left in a given program. In this case, a virus could determine the location of the needed services and hook into them, thereby adding only that code which was not already present in the host program. Even without the symbol table, a virus could search the host program, looking for the services it requires. Then, it would import only those services which it could not find.[4] Also the virus could escape detection by inserting a dummy system call that is absent from the uninfected program, pushing the system call number onto the stack and jumping to the trap instruction inserted. Such viruses would escape detection by the current detector, although it could be extended to identify code that searches a program for system calls. We are currently investigating these and other approaches to defeat the detector and to extend the detector to make it more robust.

3 The Filter

3.1 Basic Approach

A *virus filter* is an automatic classifier which applies static analysis techniques to detect the presence of a virus. Since computer viruses multiply by implanting themselves in healthy programs, a necessary condition for propagation is their ability to modify executables. Our approach, although based on the technique of formal verification differs from classical verification. Verification entails proving a program with respect to a specification – a statement of what function the program is intended to compute. For the purpose of detecting suspicious code, we are assuming no specification will be provided. Instead, programmed into the filter is a property to be determined of the program under analysis. For the current version of the filter, the property is "the files that the program could write to". The basic approach is first to identify all open calls in the program and then

[4]This may not be as easy as it sounds, however, since the virus must then know where each of its constituent parts is located within its code as well as how to extract them.

to enumerate the possible filename arguments to these calls. As we demonstrate, the analysis is feasible as only a small fraction of a program is involved in generating filenames. Upon being presented with the names of files that the program could write, the user could determine if the program is suspicious. Of course, a virus could still be present, but its propagation would be severely limited – essentially to just those files. Crocker and Pozzo (see [4]) (hereafter abbreviated to Crocker) proposed a virus filter based on formal specification and verification techniques. But through the following hypotheses, they conjecture that the analysis will be vastly simple than that usually associated with program verification.

Hypothesis 1 It is possible to formulate restrictions for the majority of useful programs such that the restriction is syntactically simple enough to be machine processable and fine-grained enough to represent the full range of authorized modifications made by real programs. A restriction is the specification of the modifications a program makes. It is created by a program developer wishing to submit an executable program for potential use.

Hypothesis 2 It is possible, on the average, to analyze benign programs in a straightforward way.

Hypothesis 3 It is possible to classify modifications such that ordinary changes can be distinguished from suspicious ones.

Generally, we agree with Crocker's hypotheses, but argue that for some programs (benign or infected) the semantic analysis required is more complicated than implied by these hypotheses.

In UNIX systems, the propagation of a virus through direct access to files is through the *open*, *create*, *rename*, *link* and *unlink* system calls. A virus may open and write to an executable or replace an executable by its viral counterpart. Using symbolic evaluation techniques, it is sometimes possible to determine the arguments to these system calls and hence the names of files being modified. The enumeration of the files which may be modified by the program being investigated provides clues to detecting viruses. For example, the program `date` does not write to any files (except standard output). If the enumerated list of files the filter identifies for `date` is not empty, it can be concluded that the `date` program is suspicious. The analysis of the benign `date` program is very straightforward. Much less straightforward is the `split` program. `Split` reads a file and writes it in n-line pieces to a set of output files. The name of the first output file is an argument specified in the command line with "aa" appended, the second one with "ab" appended, and so on. If no output file argument is given, "x" is used as default. The program should only create files starting with the prefix specified in the command line or the default prefix. Therefore, we can say the `split` program is safe if the enumerated files satisfy this restriction.

In general, a program is said to be suspicious when

1. The program's acceptance criteria is not satisfied – there is a high potential for a virus. The acceptance criteria states that the enumerated set of filenames is acceptable to the user.
2. The program is too complex to be evaluated by our filter. No definitive answer is obtained from the filter so the program is not accepted. In practice, it would be the responsibility of the programmer to argue that a suspicious program is not contaminated.

Otherwise the program is said to be safe.

After sampling some commonly-used programs, Crocker concluded that the patterns of filename generation could be classified as follows:

Implied – There is a fixed, possibly empty, list of files to be modified. For example, `date` modifies no file. `vipw` modifies `/etc/passwd`.

Parameters – Filenames are passed to the program as command line arguments. For example, indent indents and formats a C program specified in the command line.

Transformations – Some programs such as compilers and editors create new files based on the arguments in the command line. For example, `compress` transforms `filename` to `filename.Z`.

Temporary files – New filenames are generated independently. For example, `vi` generates temporary files in the `/tmp` directory.

Dialogs – The filename is provided by the user when the program is running. For example, `csh` (a standard UNIX command interpreter) file redirections are obtained from terminal input.

In all of these classifications, the algorithms used to generate filenames are quite simple involve a small fraction of the total program. Since most realistic programs are far too complex to be analyzed in their entirety and most of the code is unrelated to filename processing, our approach is to isolate that part of the program concerned with filename generation and disregard the remaining part. The simplicity of the resulting reduced program should make the static analysis tractable.

In summary, our filter tries to determine the names of all files which might be modified by the program. By comparing the enumeration of names and the specified restriction, the virus filter can claim the program is safe or is suspicious. The complexity of the programs in their entirety may prohibit comprehensive analysis, so part of our method eliminates that part of the program not related to filename processing. We call this method *slicing*. After slicing, the residual program is usually small in size and, thus, analyzable.

A virus in a program could escape detection by the filter if it is content to contaminate only those files for which the program has legitimate access. For example, a virus hiding in the EMACS editor could infect a program being created using the editor. However, once infected this program could infect only those programs its designer has given it access to. Any code in the original virus that would involve writes to other files would be detected by the filter.

3.2 Implementation and Results

This section discusses the implementation of our approach. The input to the virus filter is a binary executable. The output is the enumerated set of the files that may be modified by this executable. The virus filter proceeds through six steps. The first five steps are the preprocessings required to extract the program fragments which contribute to filename generation. The last step involves symbolic execution and analysis. The six steps are as follows:

1. Translation to an intermediate language
2. Determination of basic block and life span
3. Determination of data dependencies
4. Anti-aliasing
5. Slicing
6. Symbolic evaluation and analysis

Given a program to be analyzed, the virus filter first translates it into a C-like intermediate language. Then the filter relabels variables in order to decouple semantically disjoint variables sharing the same storage. Next, the data dependencies are found by analyzing the program syntactically. The filter performs anti-aliasing analysis to unify references to the same storage. Extra dependencies are added to the data dependency graph when aliased storage is found. Based on the data dependence graph, the program is sliced into pieces. Finally, the pieces which are related to filename processing are extracted and symbolically executed. The filter also applies some theorem proving techniques, primarily to derive inductive assertions for the few, if any, loops involved in filename enumeration.

The following simple example, written in our C-like intermediate language, is used to illustrate the different steps of the virus filter. This example program consists of two independent fragments of code which perform different operations although they share the same variables. It demonstrates our method of decoupling variables by relabeling. Then, we separate it into two independent program fragments by applying slicing. After locating the appropriate fragment containing the system calls, we apply symbolic evaluation and analysis to determine the filenames.

Example: We pick up this example after translation to an intermediate C-like language. `x` is a filename string, i is an integer, `str()` is a function converting an integer to a string. Not shown are the *open* system calls, assumed to occur at any line in the program with filename argument x.

Line number	Intermediate code	
1	`i = 1`	
2	`x = ''f''`	
3	`x = x ‖ str(i)`	# string concatenation
4	`i = i + 1`	
5	`if (i <= 3) goto line 3`	
6	`print x`	
7	`i = 200`	
8	`x = str(i)`	

The filenames generated would be:

`f`	if the open system call follows line 2
`f1, f12, f123`	follows line 5
`f123`	follows line 6
`200`	follows line 8.

3.2.1 Translation to Intermediate Language

The input to the virus filter is assumed to be a machine compiled binary program, not an arbitrary assembly language program. In the first stage, the program is decompiled into a machine independent, C-like, intermediate language. We have designed the intermediate language such that analysis attendant to steps 2-6 is simplified. To be specific, the intermediate language contains at most one assignment per statement and control is transferred by the goto statement only. The decompiler recovers semantic information about variables which are lost during the compilation. The goal is to partition memory into regions such that each region is the storage for a simple or structured variable. All storage locations are made explicit and side effects are eliminated. Library calls, like string assignments (string copy) and integer to string conversions, are replaced with defined functions in the intermediate language. Thus the virus filter is more likely to produce intelligible output through reference to higher level functions.

Since our filter is designed to work with binary executables, we need a decompiler to translate machine codes to the intermediate language. Intuitively, the intermediate language should contain more information than the machine code, e.g. concerning types and addresses of symbols. We should be able to locate extra information concerned with the high-level language from sources such as the symbol table. Even if we cannot find anything directly, we may still be able to deduce data types, procedure entries, etc, from the style in which the compiler generates code.

3.2.2 Basic Block and Life Span Analysis

Variables are often *recycled* in many programs in order to save storage or simply as a matter of programming style. In many programs, variable i is a general purpose loop counter which is reused in different, unrelated parts of the program. This recycling adds dependencies to the dataflow graph that can be eliminated. The elimination involves the renaming of the variables on the left hand side of an assignment statements.

After translating to the intermediate language and relabeling, the program is decomposed into basic blocks for life span analysis. A basic block is a sequence of instructions in which

1. All control transfer statements are at the end of the block.
2. Only the head of a basic block can be the target of any control transfer statements.

The life span of a variable corresponding to an assignment is the span of validity of its value. The life of a variable starts on its assignment and propagates to basic blocks that the current block can lead to. We now pick up the example be derived as the filter starts in step 2. In line 4 of the following table, the value of i at the right hand side may be derived from three possible sources because there are 3 assignments to i (lines 1, 4, and 7). The purpose of life span analysis is to eliminate impossible combinations, i.e. i.7 can never be the i.4 of line 4.

Variables on the left hand side are relabeled uniquely by their name and line number. The program is broken into three basic blocks. The live variables are given in the rightmost columns.

Line number	Intermediate code	Life of i	Life of x
1	`i.1 = 1`		
2	`x.2 = ''f''`	`i.1`	
3	`x.3 = x || str(i)`	`i.1 i.4`	`x.2 x.3`
4	`i.4 = i + 1`	`i.1 i.4`	`x.3`
5	`if (i < 3) goto line 3`	`i.4`	`x.3`
6	`print x`	`i.4`	`x.3`
7	`i.7 = 200`	`i.7`	`x.3`
8	`x.8 = str(i)`	`i.7`	

3.2.3 Finding Data Dependencies

Given the life span of the variables, the syntactic data dependencies can be determined by dataflow analysis. Consider, for example, statement 3 in the example after step 2: "x.3 = x || str(i)". The variables x and i are referenced; x.2 and x.3 are live when x is referenced; i.1 and i.4 are live when i is referenced; x.3 is written to. Thus x.3 depends on i.1, i.4, x.2, and x.3.

Thus for step 3, the dependencies are determined to be:

```
x.3 ← x.2, /* x.3 depends on x.2 */
x.3 ← x.3
x.3 ← i.1
x.3 ← i.4
i.4 ← i.1
i.4 ← i.4
x.8 ← i.7
```

The objective of the data dependency analysis is to slice the program into independent portions to simplify static analysis. Since filenames are usually generated by simple algorithms, syntactic dependencies are considered instead of semantic (real) dependencies. This simplified analysis is adequate for the worst case dependencies. In the above example, we can recursively trace back and find all variables on which x depends at line 8. The dependency subset is found to be "x.8 ← i.7". The subset program is composed of lines 7 and 8 as indicated by line numbers in the dependency subset.

Similarly, the variables x.2, x.3, i.1, i.4 are related to the computation of x at line 6. Lines 1 to 4 constitute the corresponding subset program.

3.2.4 Performing Anti-aliasing

We need to solve the aliasing problem which results from the possibility of referencing a memory location directly through a variable or indirectly through a pointer. Such sharing of storage must be identified before we can have a correct data dependency graph. After the virus filter identifies the aliases, additional dependency arcs are added into the graph. The aliasing is found by considering the pointer assignments. Let us call the variable on the left hand side of the assignment statement the 'home' variable. Reference through a pointer will add a dependency to this variable. Modification through the pointer will add new labels to the home variable. Since the life of the new label must be computed, the virus filter may need to iterate through steps 2 to 4 several times. The iteration stops when no new dependencies are identified.

3.2.5 Slicing

After completing steps 1 to 4, we have the data dependency graph and the next step is slicing to identify the program fragment associated with each open system call. A fragment terminates with an implied system call, the arguments of which are to be determined in step 6.

Continuing with example 1, if the system call immediately follows line 8, the sliced fragment would be:

```
7  i = 200
8  x = str(i)
```

If the system call immediately follows line 3, the slice fragment would be:

```
1  i = 1
2  x = ''f''
3  x = x || str(i)
```

To obtain the pertinent program fragment, the filter traces back from the system call through the data dependency graph to obtain all of the variables the system call depends on, i.e, the line numbers of the relevant program statements. Having the line numbers, we can easily *slice* out the program fragment.

3.2.6 Symbolic Evaluation and Analysis

The sliced program is then symbolically executed to identify filenames generated. Proceeding forward through the statements in a program fragment, each variable obtains a set of values, each value in the set being a value the variable could be assigned in a real execution.

The symbolic evaluation is straightforward when the program has no loops. For a program containing loops, more complicated techniques, as described by German and Wegbreit in [5], are required.

Given the input and output assertions for a loop, four methods to obtain inductive assertions for the program have been proposed: (1) weak interpretation, (2) using loop exit tests and generalization, (3) predicate propagation, and (4) extracting information from unsuccessful proofs. The first three methods can be used in our virus filter. The last one is not applicable because it works backward from the output assertion, which we assume will not be available.

The followings are the salient points of German and Wegbreit's first three methods as they bear on the virus filter:

1. Symbolic evaluation in a weak interpretation.

   ```
   P = start address of S;
   {I: start address of S <= P <= end address of S}
   while (P < end address of S)
          {I}
          P = P + 1;
          {I}
   ```

   For example, suppose P is a pointer variable and S is a string variable. P is initialized to the start address of S on entry to a loop; P is incremented on each pass through the loop, and the loop is exited when P is greater than the end address of S. It follows that inside the loop, the inductive assertion I will contain the expression: start address of S $\leq$ P $\leq$ end address of S. Weak interpretation attempts to derive simple facts of this kind; specifically, it considers only simple linear equalities or inequalities relating two variables.

2. Combining assertions with loop exit information.

   Suppose a loop is exited when some test D is true and that after the loop some assertion P is to hold. Since P is to hold after the loop, the assertion $D \rightarrow P$ (read D implies P) must be true inside the loop and just before the exit test. It is very likely that $D \rightarrow P$ is sufficiently strong a loop invariant for our purpose.

3. Propagating valid assertions forward through the program, modifying them as required by the program transformations.

   Whenever an assertion is known to be valid, it is useful to propagate it forward in the program, deriving the strongest consequences of the assertion downstream. Through substitutions, assertions are modified on passing through decisions and assignments to produce their consequences.

   Our preliminary analysis of the filter has determined that these 3 methods are adequate for the analysis of loops involving file enumeration code.

3.3 Example: The Split and the Copy Programs

The program split.c is analyzed. The synopsis of split is split [-number] [infile] [outfile].

In short, split reads a file and writes it in n-lines pieces onto a set of output files. The name of the first output file is an argument specified in the command line with "aa" appended, the second one is outfile with "ab" appended, and so on; the name generated form a lexicographic sequence. If no argument is given, "x" is used as default. The following is the sliced split program, resulting from applying steps 1-5 of the filter.

```
10 argc = INPUT;  argv = INPUT
16 outfile = "x"
21 for (i = 1; i < argc; i++)
38  outfile = argv[i]
42 outfile = outfile || "aa"
43 for (suffix = outfile; *suffix != 0; suffix++)
45 suffix--
47 *suffix = 'a' - 1
81 if (++*suffix > 'z')
82         *suffix = 'a'
83         ++*(suffix - 1)
87 creat(outfile, 0644)
```

The slicing reduces the 104 line program to 12 lines. As we can see, the program fragment for the generation of filenames is very small even though not trivial compared with other programs we have considered. By symbolic evaluation, and tracing through the loop several times, the result is

```
("x" | argv[*])  || ("aa" | "ab" | ... ).
```

Using German and Wegbreit's methods for the derivation of the loop invariant, we have the conditions *suffix > 'z', *(suffix+1) > 'z', *suffix = 'a' - 1, and *suffix is not decremented in the loop. From these conditions, the following represents possible value for the filenames.

```
("x" | argv[*]) || a || b.
```

where "a" $\leq$ a, b $\leq$ "z".

The user would accept split as safe, as it writes only to files that he expects.

As another example, consider 'cp' which copies files. The synopsis is

"cp filename1 filename2"

or

"cp filename ... dirname".

In the first format, cp copies filename1 to filename2. In the second format, cp copies the filename ... to the directory dirname. The sliced program fragment is like

```
39 creat(argv[2], sbuf.stmode & 0777)
70 ptr = argv[argc - 1]
71 dp = dirname
```

```
72 while (*ptr != 0) *dp++ = *ptr++
74 *dp++ = '/'
75 ptr = argv[i]
78 while (*ptr != 0) ptr++
79 while (ptr > argv[i] && *ptr != '/') ptr--
80 if (*ptr == '/') ptr++
81 while (*ptr != 0) *dp++ = *ptr++
82 *dp++ = 0
84 creat(dirname, sbuf.stmode & 0777)
```

The original program is 154 lines. lines. The sliced fragment is very small and most of the programming statements are strings operations consisting of small loops.

3.4 Limitations of the Filter

Since static analysis techniques are crucial to the operation of the virus filter program, it is assumed that the program being analyzed is constrained to *good* programming practice. That is, the code segment cannot be altered and control may not be transferred to the data segment or the stack segment. These constraints are satisfied for Sun UNIX 3.2 programs. Most programs do not change their code segment or try to execute the data segment. Dynamic linking programs and debuggers are exceptions, albeit important ones. Furthermore, our model assumes a single thread of control. Modifications would be required for a parallel program with shared memory. Some specific limitations of the current virus filter design are discussed below.

3.4.1 Pointer reference

There is no constraint on indirect memory references (any pointer or array reference) in our low-level language. Whenever this kind of reference is associated with a string of unknown length or with a non-deterministic event (e.g. a user input), any memory cell can potentially be modified. The overwritten cell can be anywhere, possibly a filename argument to system calls.

If the symbolic evaluator works conservatively, its output will be too pessimistic, as most storage contains non-determined values. Otherwise the output of the evaluator could be incorrect and a virus could slip into the system without being detected.

The following are common statements in C programs; see Figure 1.

```
while (*p++ = *q++);                  -----------
                                      |   str   |<------- p
--------------------                  |         |
                                      |---------|
scanf(''\%d'', &i);                   |         |
str[i] = 'x';                         |  file-  |
                                      |  name   |
                                      -----------
```

Figure 1. Two examples to illustrate the difficulty of deciding pointer references.

In the first example, if p points initially to a variable 'str' and memory is allocated as shown, p can overflow and, potentially, clobber storage that holds the filename argument to a future system call. We potentially have to determine all possible values of pointers in order to determine what storage can be clobbered.

The undisciplined use of pointers severely hampers effective static analysis, but also represents bad programming style. In the first example, a better alternative is to use

```
strncpy(p, q, MAXLEN).
```

The second example should have a conditional statement such as

```
if ((i < 0) || (i >= UPPER_BOUND)) input_error();
```

before the array reference to avoid an out-of-bound array reference. Assuming these extra statements, it is possible to bound the pointer references to facilitate analysis.

We may be able to infer the range of pointers from the program and prove the pointers are constrained with their associated variable. For example, during the life of a pointer P associated with string S, our filter proves the assertion $\{S \leq P \leq S + \text{length}(S)\}$.

Another difficulty with pointers is with respect to dynamic allocation of memory. It is almost impossible to find the bounds of pointers pointing to dynamically allocated memory variables because there is no way to determine their addresses statically. If we assume the worst case – all pointers can share all dynamically allocated storage – it is not likely that the filter can perform an effective analysis of the program.

Although not acceptable in all situations, it appears that software users can impose a strict programming style on their vendors to simplify the work of the virus filter in pointer analysis.

3.4.2 Loops

As discussed before, the presence of loops makes symbolic evaluation much more complicated because of the indefinite number of iterations. Several methods may suggest solutions to this problem. One of these is to determine the maximum number of iterations of the loop and have the symbolic evaluator go through the loop that number of times. Other methods include the determination of loop invariants to give significant representation for the loop. A method which uses linear inequalities to constrain the variables may be useful [5]. However, all existing solutions are heuristics and do not work in all situations.

3.4.3 Structured Data Types

The symbolic evaluator needs to understand structured data such as strings and records. String operations are common in generating filenames. The evaluator should be able to understand that the code fragment *while (*p++)* is equivalent to moving the character pointer p to the end of the string. Some heuristics are helpful in giving understandable reports to a user. For instance, it is preferable for the filter to output the statement, "A new string S1 is generated from S by appending character 'c' to it" rather than output the assertion

$$\begin{aligned}&\{\ \exists j((\forall i < j : S[i] != 0 \text{ and } S[j] == 0) \text{ and}\\&(\forall i < j : S1[i] = S[i]) \text{ and}\\&S1[j] =' c' \text{ and}\\&S1[j+1] = 0)\ \}.\end{aligned}$$

In some situations, the complete filenames may not be generable in the absence of specific details of the environment. For example, when a temporary file is generated with the constant prefix */tmp/vi* and the process-id, the symbolic evaluator is unable to give the exact filename because the filename depends on the runtime environment. However, if the evaluator is sufficiently intelligent, it may give a partial result such as */tmp/vi** as the generated filename. However, the cost of having such intelligence is not low as the evaluator needs to understand the semantics of strings and essentially all possible operations on a string.

4 Conclusions and Future Work

There is a need for improved defenses against computer viruses. Defenses include

1. preventing the propagation of viruses
2. detecting an infected program
3. determining if a newly issued program contains a virus.

This paper concentrates on (2) and (3). Our detector tool checks for duplication of services. A program linked with the standard library, typical of UNIX systems, will contain no duplication of system services. A simple virus would carry its own services and would be easily detected by our detector tool. The detector can be defeated, but only by a virus that searches for the services in a program; such a virus will be more complex than current viruses and might contain code that an extension of the detector would flag as malicious.

The filter tool extends the concepts proposed by Crocker and Pozzo. It carries out a static analysis of a given program to determine the capability of a program to modify files. A user of the program under test could then determine if any unexpected files are written to, for example those obtained by searching a directory. The filter uses verification techniques, but since only a subset of a program is usually concerned with filename generation, the technique appears to be more feasible than verification in general. We have simulated the behavior of the filter on typical system programs, such as `date`, `split` and cp. Heuristics are required to generate loop invariants (but the loops appear to be quite simple) and to demonstrate that pointers are well-behaved. Implementation is underway.

The prototype of the virus filter will be tested on MINIX system utilities executing on a 80286-based machine. MINIX is a UNIX-like operating system written by Tanenbaum [9]. The MINIX programs are usually small, making them ideal for an initial evaluation of the virus filter. Also, the assembly language is quite simple, which will simplify the translation to the intermediate language.

The similarity between the detector tool and the filter tool is that both attempt to determine if a program under test contains suspicious code. The difference lies in the suspicious code under searching. The detector tool considers program structure, i.e. the way that system calls are made. The filter focuses on the arguments to the system calls. System calls are interesting because a program may interact with other objects in the system, hence cause damage, only through operating system calls.

Generalizations of the detector would involve more complex checks on program structure. For example, the detector might look for the getdirentries (get directory entries) system call which is useful to viruses, but not to most programs. Different compilers generate code in slightly different ways. If the virus code is compiled with a foreign compiler, the detector may be able to detect it with statistical or pattern-matching methods. Furthermore, it is common for a virus to attach itself to the beginning or the end of a program. By looking at the pattern of flow controls, we may obtain some hints to the presence of viruses.

The filter tool uses symbolic evaluation and verification to determine the possible arguments of the system calls. It can be extended to determine values of variables in the program; hence we can prove assertions composed of program variables, which characterize the program behavior. The filter may determine the input conditions which lead to execution of certain sections in the program. For example, if we apply this technique to the *login* program and ask for the condition that the *setuid* statement is executed, the filter should find that a necessary condition is the matching of passwords.

The techniques described in this paper are not limited to the detection of viruses. Trojan Horses are detected in similar way, albeit the detector and filter need to be programmed with different properties.

Because virus detection is undecidable (see [1]), these tools certainly cannot claim to have the ability to detect all viruses. The detector tool can be defeated in at least one sure way by using the existing services of a program. Similarly, the filter can also be defeated: a virus can propagate through the files to which the program has legitimate access. Although these tools cannot detect all viruses, viruses have to hide all the traits we are looking for. A virus designed with these constraints is very complicated, cannot be very infective, and also very hard to write. The mere complexity of such a virus should lead to its early discovery by more common methods of debugging.

5 Acknowledgments

John M. Collins of St. Albans, Herts, England, wrote the disassembler used in Snitch. We are also grateful to Steve Crocker, Maria Pozzo, Doug Mansur and his colleagues at the Lawerence Livermore National Laboratory for many conversations on the virus problem.

References

[1] Fred Cohen. "Computer Viruses: Theory and Experiments", *Computer Security: A Global Challenge*, J.H. Finch and E.G. Dougall (eds.) (1984)

[2] Fred Cohen. "A Cryptographic Checksum for Integrity Protection", *Computers & Security*, Vol. 6 pp. 505-510 (1987)

[3] Fred Cohen. "Models of Practical Defenses Against Computer Viruses", *Computers & Security*, Vol. 8 pp. 149-160 (1989)

[4] Steve Crocker and Maria M. Pozzo. "A Proposal for a Verification-Based Virus Filter", *Proc. of the 1989 IEEE Computer Society Symposium on Security and Privacy*, May 1-3, Oakland, California, pp. 319-324 (1989)

[5] Steven M. German and Ben Wegbreit. "A Synthesizer of Inductive Assertions," *IEEE Trans. on Software Engineering*, Vol. SE-1, No. 1 (1975)

[6] Mark K. Joseph and Algirdas Avizienis. "A Fault Tolerance Approach to Computer Viruses", *Proc. IEEE*, pp.52-58 (1988)

[7] Aamer Mahmood and E. J. McCluskey. "Concurrent Error Detection Using Watchdog Processors–A Survey", *IEEE Transactions on Computers*, Vol. 37, No. 2 pp. 160-174 (1988)

[8] Maria M. Pozzo and Terence E. Gray. "An Approach to Containing Computer Viruses", *Computers & Security*, Vol.6 pp. 321-331 (1987)

[9] Andrew Tanenbaum. *Operating Systems: Design and Implementation*, Englewood Cliffs, N.J., Prentice-Hall, Inc. (1987)

[10] Paul A. Karger, "Limiting the Damage Potential of Discretionary Trojan Horses", *Proc. of the 1987 IEEE Computer Society Symposium on Security and Privacy*, Oakland, California, pp. 32-37 (1987)

[11] D.R. Wichers, D.M. Cook, R.A. Olsson, J. Crossley, P. Kerchen, K.N. Levitt, R. Lo. "PACL's: An Access Control List Approach to Anti-Viral Security", *to appear in Proc. of the National Computer Security Conference, 1990.*

The Virus Intervention & Control Experiment (VICE)

by James E. Molini & Chris W. Ruhl
Computer Sciences Corporation
16511 Space Center Blvd.
Houston, TX 77058

Prepared under Contract # NAS9-17920

ABSTRACT

This paper describes a program recently instituted at NASA's Johnson Space Center to reduce the impact of Personal Computer Viruses. The program attempts to provide successive levels of detection and control for virus programs in a cost effective and consistent manner. This description includes design notes and an outline of problems encountered while planning the implementation.

INTRODUCTION

During 1989 the NASA Johnson space Center (JSC) had approximately 100 suspected virus infections. Of those suspected infections only 30 proved to be actual viruses. Of the 30 actual viruses handled in 1989, only one MS-DOS compatible machine was infected.

During that same year there were over 5000 PC's owned by the U.S. Government and thousands more owned, or leased by local contractors. Over 90% of those PC's were MS-DOS compatible systems. The PC's were connected to virtually every major scientific research facility in the U.S. and most were connected to a wide area network that served more than 200,000 NASA users worldwide.

This information brings viruses into a more appropriate perspective. It also belies the fact that hundreds of employee hours were spent checking for non-existent viruses during that year. After the Datacrime Virus scare of 1989 many of us at the center began working on possible solutions to the problem. The solution is officially named the JSC Virus Intervention/Disinfection Program. This paper explores the mechanism of that solution.

BACKGROUND

The experience of supporting several thousand users in a communication intensive environment forces managers to place a strong emphasis on practical, cost effective solutions to problems. For 98% of our PC's, the loss of any 1, or 2 machines would not seriously affect center operations. Therefore, the idea of preventing, or detecting, every virus on every machine became a tradeoff between the time saved by the solution and the time spent on the problem. In this regard it became obvious that isolation, user convenience, and transparency were more important than 100% prevention. Going into this effort we understood that viruses were not completely preventable on the target platform. So our exploration focused on providing a reasonable level of detection and isolation for center PC's in accordance with the risk anticipated over the next 2 years.

We initially attempted to identify a commercial product which would satisfy our objectives. After an extensive search, we eliminated all candidates because they failed in at least one of the following areas:

1. The cost to procure and implement exceeded our available discretionary funding limit of $50,000.

2. The software required frequent updates to remain effective.

3. The software was not simple, or user-friendly.

4. The software would not identify all existing boot infector, or file infector viruses. (Failure to identify all existing viruses would seriously impair user trust over the next 2 years.)

5. The software was not resistant to a directed virus, or Trojan Horse. With file comparison programs, this usually stemmed from a lack of sufficient granularity, or sophistication during the scan.

Thus, although we had not intended to become developers, we were forced to explore novel approaches to the problem.

PROJECT DESIGN AND OBJECTIVES

One of the authors had previously been using a public domain program called FILETEST, which used Cyclic Redundancy Checks (CRC's) to check for file modifications. Using this as a starting point, we proposed modifying FILETEST and using the modified program as a virus detector on MS-DOS based machines at the center. Using an integrity check utility as the first line of defense, we would then be able to quickly identify problems. Once the existence of a problem had been detected, we would then be able to more efficiently employ our other virus response techniques in_the repair.

Although we had also proposed a hard disk write protection utility, this concept was omitted from the task. It was suggested by management that write protecting hard disks might be too complicated an option for novice users. Later testing proved this point to be true. After several initial meetings with user representatives and JSC management we refined the proposal above into a single goal statement. The initial goal statement for the project was:

> Perform an integrity check of executable files on user hard disks on a routine basis to identify all modifications. Provide users with appropriate instructions when changes are noted.

Based upon this goal, a utility was designed to meet the following requirements:

1. The utility should be a first line of defense. It does not need to be a "Universal Virus Detector," but should give all users a reasonable amount of assurance that their machines are clean.

2. It should work without needing updates for extended periods of time. Updates not more than every 2-3 years would be desirable.

3. The detection program should be simple and provide user-friendly feedback on what it finds. It should not confuse the inexperienced user.

4. An unskilled user should not be able to do more harm than good when using this utility.

5. The utility should significantly reduce the number of unnecessary virus response visits made by Help Desk personnel. (ie. User alerts should provide enough information so that false alarms can be debugged over the phone.)

6. The finished product should be placed into the public domain so that other NASA centers or contractors may use it without charge or license fees.

7. Trained personnel will install the utility. Therefore, detailed installation instructions will not be required.

At the preliminary design meeting we added the following criteria to our design:

1. The utility should perform a static analysis of executable files. Too many potential memory conflicts would result if the program were installed as a Terminate & Stay Resident (TSR) routine.

2. The program should become an integral part of the menu utility. This means that whenever the program is installed, it should be installed along with a new version of the menu. As such, it will become a permanent part of "extended operating system."

SYSTEM INTEGRITY CHECK

The System Integrity Check utility has a long history, which must be noted in order to give credit to the appropriate parties. Dr. Ted H. Emigh initially wrote a public domain program called FILECRC (4), using a set of CRC routines written by David Dantowitz. The CRC's implemented are based upon 16 bit polynomials, one of which is a CCITT standard CRC polynomial. They were chosen because of their ability to detect significant bit errors in data streams.

The standard CCITT Cyclic Redundancy Check is mathematically represented by the equation: CRC-CCITT = X^16 + X^12 + X^5 + 1. This 16 bit CRC will catch all 16 bit bursts[1], a high percentage of random 17 bit bursts (~99.997%) and also a large percentage of random 18 bit or larger bursts (~99.998%).(4)

The FILECRC program was then modified by Dr. Leonard P. Levine of the University of Wisconsin and placed into the public domain. Dr. Levine wrote his program, called <u>FILETEST</u> (6) in Turbo Pascal 3.0.

To make <u>FILETEST</u> more efficient and take advantage of Turbo Pascal, we rewrote the program in Turbo Pascal version 5.5 before proceeding further. This conversion caused program performance to improve by an average of 40%.

However, while using <u>FILETEST</u>, we discovered a virus that completely escaped detection by the package. The specific problem surrounded the 4096 virus, which replaced a part of DOS and masked infections dynamically during the read process. In this situation, a simple read operation failed to identify the infection. Since our design goal dictated a utility that could detect all existing viruses, we began examining ways to enhance our derivation of the program.

In order to combat this general type of virus, we examined several alternatives, such as reading to the end of each physical disk cluster, or timing known operations. Further examination showed these operations insufficient to combat future infections.

Then Chris Ruhl defined a logical model for a robust virus detection mechanism. After several weeks of work, the model was refined into its

[1] A burst of length N is defined as a sequence of N bits, where the first and last bits are incorrect and the bits in the middle are any possible combination of correct and incorrect. From (1) Peterson and Brown.

present state. (We will briefly describe the model here. A complete description of the model is available from the authors.)

VIRUS DETECTION MODEL

Assuming that [a] a virus must modify existing executable files on a computer system (any program that spreads by itself is a "worm," and a malicious program that does not spread is a Trojan Horse) and [b] that the initial state of the machine does not contain a virus, we can assert that:

1. A virus infection occurs when an executable resource on the system (in primary, or secondary storage) is altered.

Consequence: If we have a baseline configuration to work from (assumption b) then we can compare this baseline against future states. In this way, a CRC, or other polynomial manipulation of file data can be used to uniquely identify the state of the file. If the file is ever changed, this computational method can detect the change. Once the change is detected, the user must determine if the change was authorized (ie. software installation, or upgrade) or not.

2. Masking occurs when a scanning entity is prevented from seeing the change in an executable resource. Masking requires external intervention at some point in the read process. Therefore, if a program has not intervened in the read process, masking cannot occur.

Consequence: Infections on disk can be masked by an intelligent virus, but RAM based infections cannot be masked unless the operating system intervenes in a fetch or jump (ie. address translation etc.). Since a program can only execute from main memory, the masking code must reside somewhere in main memory to execute. This means that if we can effectively compare the memory-based read procedure against an uninfected state we will be able to detect masking in all cases.

3. If masking has not occurred, the only way for a virus to escape detection is to interfere with the detection program itself and/or its reference data.

Consequence: A directed virus could update either the detection program or its base tables. Subsequently, it is important to validate the integrity of the program and either protect, or authenticate the reference data used by the utility.

General Consequence: A carefully written CRC based program can detect all known viruses and validate the integrity of the operating system under which it operates. Although it is still vulnerable to certain directed attacks on specific systems, it should provide a high level of reliability across a variety of platforms.

As rewritten, the System Integrity Check provides a range of system checks based upon user input. Our derivation of the program is called DETECT. For the purposes of this paper we may use the names "Detect" and "System Integrity Check" interchangeably.

PROGRAM OPERATION

Detect can perform either a limited check, or an extended check of a target system. We will outline those checks here.

During installation of the program, an extended check is performed. All executable files, and the boot sector on the default disk are checked. This then becomes the master list that is used to generate the master

CRC table. A baseline set of CRC's is computed for this master table which encompasses every potentially executable file on the hard disk. This baseline must be computed from a clean machine. Therefore we have set up an installation process that scans the machine prior to initial installation.

Detect will also create a configuration file called USER.CFG. The user will be instructed that he/she may edit the table to include only the most commonly used files. Once this is done, the table will be used for each limited check. When the program is run using this table, any files not found in the USER.CFG file, or on the disk will be flagged with a 1 line entry such as:

```
 File C:\NORTON\SD.EXE         Removed from Detect file.
or
 File C:\NORTON\SD.EXE         Not Found on Disk Drive.
```

Files that are changed will register in one of 2 ways. Any file that was updated using normal DOS calls (eg. through recompilation, or copying in a new file over a file with the same name) will be flagged as normal updates in the following manner:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 File C:\COMMAND.COM Modified Normally.
     Old Date  =  7/24/1987     New Date    =  1/17/1990
     Old Time  =  0: 1: 2       New Time    =  7:27: 2
     Old Size  =   25276       New Size   =   25276
     Old Attr  =          0        New Attr    =       32
     Old CRC (1)=  -31695          New CRC (1)=  -16742
     Old CRC (2)=    -4475         New CRC (2)=   21008

 If you did not make this change, please contact the User Support Desk
for assistance.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

In this case users are instructed to contact the help desk if they do not recall updating the file. This message is designed to avoid causing panic in a novice user who has just received an update to his/her word processor.

If, however, the directory entries match and the CRC's do not, the file has been modified in a non-standard way. This should not occur in normal practice, so the program writes a special alarm message to the terminal. This can occur when using NORTON UTILITIES, or other such programs to modify the disk directly, bypassing the normal DOS handling of the files. It is also a common method used by virus programs to infect other executables.

If this occurs the following alarm message will appear:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 File C:\COMMAND.COM Modified   Abnormally!  <<<<<<
     Old Date  =  1/17/1990     New Date    =  1/17/1990
     Old Time  =  7:27: 2       New Time    =  7:27: 2
     Old Size  =   25276       New Size   =   25276
     Old Attr  =         32        New Attr    =       32
     Old CRC (1)=  -31695          New CRC (1)=    7054
     Old CRC (2)=    -4475         New CRC (2)=  -25852

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Virus Alert! File(s) have been Abnormally Modified  %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
This could be serious! Please call the User Support Desk at ___-____.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

In any event, at the end of each scan a summary message will also appear, displaying the number of files scanned and their problems.

Finally, the user is prompted to update all changes.

During program initialization, Detect will check the date stamp of the last full disk scan. If the date is more than 30 days old, the program will execute an extended scan of the system again. This is designed to do two things:

1. The check is designed to catch illicit modifications to little-used files and to find new executables added to the system. By doing so on a monthly basis, all files are regularly checked, but the user is spared a lengthy wait for the daily checks.

2. The check will display a summary of all files modified, added, or deleted in the past month. This record can be saved for future reference.

When the extended check is run, all deleted files are listed, but files which are added are listed also.

Some of the special features of this program are:

1. It computes 2 different CRC's for each file. Using 2 CRC's together significantly reduces the chances that any virus will infect the file without being discovered.

2. It checks the boot sector and partition table, thus showing infections by boot infector viruses.

3. During initialization, the program validates the memory based read procedure, thus detecting viruses which attempt to modify DOS services. This prevents masking by the 4096 virus and other viruses of its type, while allowing the user to confidently run the program from the hard disk.

4. It can be run when the disk is write protected, allowing users to run the program while using a disk write protect utility. A special feature in the program will check the return code from the first write attempt and notify the user that updating the tables will be prevented.

5. It occupies an average of 60K bytes on a hard disk, allowing installation on disks where space is critical.

6. It has been tested to work on a variety of platforms. We have found that the utility can check 1 MB of executable code per minute on an AT class machine and 11 MB per minute on a 80486 system. We have also eliminated false alarms under most supported packages.

DESIGN TRADEOFFS

As with the design of any program, many options were available to us. As the design progressed, we were repeatedly faced with choices between user friendly operation and security enhancements. The principal goals in this development were to make the implementation resistant to directed viruses while finding the simplest implementation for the unskilled user.

One example of the tradeoffs concerned possible restoration of the boot sector. Because of the way we checked the boot sector, we found it possible to restore sector 0 if it had been modified in some way. Then if the user found a problem with modifications, he/she could simply rewrite the old boot sector back to its original location.

Before making the change we realized that, all disk integrity questions aside, we would be generating a separate security exposure. If a restore option had been added, a penetrator could then modify the file where the boot sector was stored and harmlessly change a byte in the boot sector. In this way, a restoration of the boot sector would actually cause an undetectable infection. For this reason we avoided any attempts at fixing modified files.

Another design question revolved around the execution of the program. Technically, Detect would have been more secure if it had run from a bootable floppy disk. When booting from a floppy we had a significantly higher level of control over the execution environment. Even so, it was determined, during the design phase, that ease of use was more important than a completely secure execution environment. Therefore the program was designed to run from the hard drive, although it could reside in virtually any directory. In this way directory hiding and name changes could increase the program's resistance to directed attacks.

While the original program has been designed to run from the hard drive, we found it necessary to create another version that runs from a bootable floppy. This version supports network servers and non-standard hardware or DOS types. When this version is released, it should achieve a much higher level of detection integrity than the original program. We have requested approval to place this version into the public domain.

IMPLEMENTATION

Once the program had been developed and tested, implementation became the major concern. The logistics surrounding installation of any software package on over 5000 PC's can be staggering. Fortunately the high visibility of recent virus infections has resulted in a great deal of management support for virus detection and control.

We knew that sending out service personnel to install the utility would cost more than $50,000 by itself. (Typically a service organization must plan on 3-5 working hours to support each service call.) We proposed that the most efficient implementation involved the use of dedicated teams of installers who worked building by building after hours. In this way the time required to install the utility could be reduced to less than 30 minutes per PC.

We developed the following plan for installation:

> Installation will proceed according to office areas in specific buildings, just as carpet cleaning, or window washing is done. Notification will be handled by designated representatives of each organization. After users have been notified and the schedule is set, the team can move through in a coordinated and supervised fashion. Users who wish to be present during the installation may either lock their doors, or post a notice on the PC. In these cases, the room number will be noted and reported to the central supervisor. Using these reports, a later installation can be coordinated.
>
> During installation a team member will first scan the hard disk for viruses using a commercial package. If an infection is found it will be corrected before proceeding. Next, the utilities will be installed along with the updated menu system and a baseline set of CRC's will be generated. After this process is complete, the installer will leave an informational brochure at the machine and proceed to the next machine. If

this approach accomplishes installation for 90% of the PC's at the center, the rest can be handled on a case-by-case basis.

PACKAGE SUPPORT

Once the package is installed, other implementation concerns will arise. If a user detects a potential virus, the local help desk must be able to dispatch trained personnel to support an identification/disinfection of the virus. We have a central user support function that is able to provide this type of assistance to local users. Additionally, we are currently evaluating various commercial packages to support virus identification and removal. Copies of the package will be maintained by designated personnel who can periodically be trained in identification and disinfection techniques.

SUMMARY

In order to allay user fears and to respond quickly to possible virus infections, the Johnson Space Center has begun a comprehensive program to detect and control PC-based viruses. As part of that program, all users are being provided with free utilities to detect possible infections. If a possible infection is detected, there are several levels of support that can be provided to that user. Although the program is still in its initial stages, response has been universally good. We anticipate that we will soon have a low-cost and comprehensive method for limiting the impacts of computer viruses in a large, complex organization.

REFERENCES

1. "Cyclic Codes for Error Detection", W. W. Peterson and D. T. Brown, Proceedings of the IEEE, volume 49, pp 228-235, January 1961.

2. "A Cyclic Redundancy Checking (CRC) Algorithm" A. B. Marton and T. K. Frambs, The Honeywell Computer Journal, volume 5, number 3, 1971.

3. "Computer Networks", Andrew S. Tanenbaum, Prentice Hall, Inc., Englewood Cliffs, NJ 1981.

4. FILECRC.SRC ver. 5.2, program by Dr. Ted H. Emigh, 1988.

5. The MS-DOS Encyclopedia, Edited by Ray Duncan, Microsoft Press, Redmond, WA. 1988.

6. FILETEST, program by Leonard P. Levine, 1988.

Classification of Computer Anomalies

Klaus Brunnstein, Simone Fischer-Hübner, Morton Swimmer

Virus Test Center (VTC), Faculty of Computer Science
University of Hamburg
Schlüterstr. 70, D-2000 Hamburg 13
Telephone: +49-40-4123-4162
e-mail: brunnstein@rz.informatik.uni-hamburg.dbp.de
sfh@rz.informatik.uni-hamburg.dbp.de
swimmer@fbihh.informatik.uni-hamburg.de

Keywords: computer virus classification, virus catalog, threat description language (TDL)

Abstract: *As the viral property is generally undecidable, effective protection from special anomalies requires detailed information about all known viruses. In this paper, different approaches to classifying computer anomalies are introduced and compared. The VTC virus catalog in its current version and new efforts of its improvement by constructing a machine readable virus catalog to allow automatic information retrieval and by defining a threat description language for an expert system,are described in more detail.*

1. Introduction:

The number of known viruses has been growing rapidly. We know of about 200 viruses, of which over 150 are on IBM-PC's, over 70 on Amiga, at least 25 on Atari, and over 25 on MacIntosh (in mid 1990). Extrapolating the development, by 1994/95 at the latest more than 1000 computer viruses will be known. Whereas early viruses were rather simply programed, easy to detect with gamelike character, recent viruses show more professional techniques and malicious intent. They have become more difficult to detect and often do serious damage. Moreover, viruses are apparently beginning to be used as (D-) weapons in criminal attacks against commercial or political institutions. To deal with this problem, the computer user must have a minimum knowledge to be able to determine whether anomalous behaviour can be attributed to a virus. Information sources containing computer virus classification are needed as early warning

The computer virus catalog comprises of such information necessary to identify a known virus. It was developed at the Virus Test Center, together with the valuable help of D. Ferbrache, C. Fischer, Y. Radai, and F. Skulason. It deliberately does not contain enough details of virus programming techniques to be of much use for virus programmers, thus rigorously following the IFIP decision that strongly advises against the publishing of virus code.

2. Terminology

There is still much confusion about the exact definitions of the various computer anomalies, even amongst computer specialists. This has led to many misunderstandings. To this day there are no generally accepted definitions of all anomalies. Although, a very important part of the classification process, an extensive discussion of them would go beyond the scope of this paper. We have provided definitions that are the lowest common denominator.

In the following definitions we use the term 'routine' to mean a non-autonomous set of instructions capable of being executed on some platform, eg. a machine or an interpreter. A 'program' is an autonomous set of routines.

Def: A **Virus** is a non-autonomous set of routines that is capable of modifying programs or systems so that they contain executable copies of itself.

A virus may contain a routine that can perform a malicious function, known as the damage, unrelated to the function of the actual virus.

We have had some criticism for insisting on viruses to be non-autonomous. There are real viruses that do not need any other program to run, as they contain start-up code, or have been designed to cope without a host. We think that the explicite or implicite start-up code is a form of host, which allows our definition to hold for this exception.

Def: A **Trojan Horse** is an autonomous program that performs a harmless function, but also contains a hidden function, often destructive, unknown to the user but intentionally implemented.

Technically speaking, any program becomes a trojan horse if it becomes infected with a virus.

Many people define a Trojan Horse to be any malicious routine. We recommend rereading Homer's 'Illiad', for there it is clear that the wooden horse was the 'Trojan Horse' and not the warriors inside.

Def: A **Worm** is a set of programs or routines, that are capable of independently, or with the help of an unsuspecting user, propagating throughout a network.

Although a worm is arguably a virus, in that it reproduces itself, the difference lies in its capability to propagate over networks and that it is (in most cases) an autonomous (set of) program(s). Likewise, many viruses (esp. in PC networks) can spread over networks, but have not been explicitly programed to do so and are therefore not worms.

In the mathematical sense of the term virus (according to [Cohen 86]), our definition of virus and worm are variations of the same thing. Unfortunately, the mathematical definition also includes such MS-DOS programs like DISKCOPY (as Fred Cohen points out himself in 'Virus Bulletin'). The technical implementation of Worms and Viruses are very different, and these differences are what we are most interrested in.

There are many other 'anomalies', such as **Trapdoors, Moles, Timebombs, Logicbombs**, etc. It is often useful to talk of, for example, a timebomb being transported by a virus or a worm. These anomalies can however be autonomous programs, with, for example, the task of preventing detection by monitoring system usage of the operator (in this case a 'mole'). Trapdoors, albeit primitive ones, are being increasingly used to make the disassembly of viruses more difficult.

3. Current classification attempts

The problem of identifying a virus has been proven to be undecidable in the most general case [Cohen 86]. Current virus identification programs can and have been bypassed, and in general one must say ultimately that any protection scheme can be bypassed. For example, the recently discovered "4096" or "Frodo" virus intercepts each read access and if an infected program is read, it is restored (disinfected) on the fly, so that the user only sees a clean version of the program. This practice is also common amongst some boot sector viruses.

As a result of the media-hype, malfunctions in software or hardware are very often attributed falsely to viruses. Extensive classification is necessary, so that even an average user can quickly identify an anomaly as (or better: as not) a virus.

As the number of computer viruses grew, the necessity for detailed information became apparent. Various lists of known viruses and trojan horses were published in the past, often on electronic bulletin board systems. Initially called the 'Dirty Dozen' (counting only the strains of viruses), these lists soon became known as the 'Terrible Twenty (plus 3)' [RADAI 89a] and finally escalating to be called the 'Threatening Thirty' [RADAI 89b], and perhaps the next version will be called the 'Filthy Fifty'. Meanwhile we have reached approximately 300 viruses, counting all variations and systems. The most prominent lists are presently published by John McAfee, David Chess, Yisrael Radai, Patricia Hoffmann, Virus Bulletin and the Virus Test Center.

3.1. McAfee/David Chess

John McAfee uses the table format designed by David Chess. It describes primarily the scope and method of infection and the type of damage the virus does [McAfee 90].

McAfee describes the various attributes a virus may have as: self-encryption, memory resident, infects COMMAND.COM, infects .COM files, infects .EXE files, infects overlay files, infects floppy disks, infects boot sectors, infects partition sector table, and length of virus. In addition the following damage is possible: corrupts boot sector, affects run-time operation, corrupts executable files, corrupts data files, makes part/all of disk unusable, and corrupts file linkage.

The following is an extract from John McAfee's Virus Characteristics List (Version_57):

```
            VIRUS CHARACTERISTICS LIST  V57
                                     Copyright 1989, McAfee Associates
                                                     408 988 3832

      The following list outlines the critical characteristics of the known
IBM PC and compatible viruses.

==================================================================
Infects Fixed Disk Partition Table---------------+
Infects Fixed Disk Boot Sector-----------------+ |
Infects Floppy Diskette Boot ----------------+ | |
Infects Overlay Files----------------------+ | | |
Infects EXE Files------------------------+ | | | |
Infects COM files----------------------+ | | | | |
Infects COMMAND.COM------------------+ | | | | | |
Virus Remains Resident-------------+ | | | | | | |
Virus Uses Self-Encryption-------+ | | | | | | | |
                                 | | | | | | | | |
                                 | | | | | | | | | Increase in
                                 | | | | | | | | |  Infected
                                 | | | | | | | | |  Program's
                                 | | | | | | | | |    Size
                                 | | | | | | | | |      |
Virus               Disinfector  V V V V V V V V V      V    Damage
--------------------------------------------------------------------
Ping Pong-B            CleanUp   . x . . . . x x .     N/A      O,B
Lehigh                 CleanUp   . x x . . . . . . Overwrites   P,F
Vienna/648             M-VIENNA  . . . x . . . . .     648      P
Jerusalem-B            CleanUp   . x . x x x . . .     1808     O,P
Jerusalem              CleanUp   . x . x x x . . .     1808     O,P

Legend:

Damage Fields -   B - Corrupts or overwrites Boot Sector
                        O - Affects system run-time operation
                        P - Corrupts program or overlay files
                        D - Corrupts data files
                        F - Formats or erases all/part of disk
                        L - Directly or indirectly corrupts file linkage

Size Increase -   The length, in bytes, by which an infected
                  program or overlay file will increase

Characteristics - x - Yes
                  . - No

Disinfectors -    SCAN/D       - VIRUSCAN with /D option
                  SCAN/D/A     - VIRUSCAN with /D and /A options
                  MDISK/P      - MDISK with "P" option
                  All Others - The name of disinfecting program
```

3.2. Radai

Yisrael Radai also uses a table format for classifying the viruses [Radai 89a][Radai 89b]. He lists the names and aliases of each strain, the number of viruses in the strain, the type of virus, including possible lengths and the date of first appearance.

The following is an extract from Yisrael Radai's PC/MS-DOS Virus List (October '89):

```
                                 No. of                                First
    Names                        Strains   Type                        Appearance
    -----                        -------   ----                        ----------
4.  Lehigh                       2         COMMAND.COM RO 0            Nov 87
5.  Vienna, Austrian
    DOS-62, Unesco               3         COM D 648                   Dec? 87
6.  Israeli, Friday-13,
    Jerusalem                    12        COM/EXE R 1813/1808         Dec 87
9.  Ping Pong, Bouncing-
    Ball, Italian                3         Boot sector 2K              Mar 88
```

Yisrael Radai's virus list is useful as it lists viruses by strain. As most viruses in a strain have similar characteristics this makes the list easier to read.

3.3. Virus Bulletin

Virus Bulletin is published in the U.K. by Edward Wilding. It contains written articles on various viruses as well as lists of known viruses. In recent issues the information given has been restricted to search string and lengths of the viruses, with to occasional added information. They regretably parted from describing the viruses in greater detail (with approximately the same scope as Radai) early on.

3.4. Patricia M. Hoffman: Virus Information Summary List

This year Patricia Hoffmann published the "Virus Information Summary List" that contains information that she has collected on over 70 MS-DOS viruses [Hoffmann 90]. As she says in the introduction, it is not intended to provide a very technical description, but to show what the virus generally does, how it activates and how to get rid of it.

The current summary list contains the following entry fields:

- Virus Name
- Aliases
- Effective Length
- Type Code(s): Following codes to indicate general behavior characteristics:
  A=Infects all program files (COM&EXE), B= Boot virus, C= Infects COM-Files only, D= Infects DOS boot sector on hard disks, E= Infects EXE files only, F= Floppy (360K) only, K= Infects COMMAND.COM, M= Infects Master boot sector on hard disk, N= Non resident, O= Overwriting virus, P=Parasitic virus, R= Resident, S=Swapping virus, T= Manipulation of the FAT, X= Manipulation,/Infection of the Partition Table.
- Detection Method: e.g. available detecting programs (Skulason's F-PROT, IBM Scan, and McAfee's Pro-Scan, ViruScan are referenced in the entries).
- Removal Instructions: e.g. availble virus disinfectors
- General Comments.

3.5. VTC Virus catalog 1.2

The **Computer Virus Catalog** (developed first in October 1988) was intended to describe the viruses in enough detail so that a user can positively identify a possible virus attack (see Appendix 1). The catalog distinguishes between the most prominent features of viruses and leaves room for finer technical detail to be given. All catalog entries provide technical information obtained by reverse engineering and analysis of the virus by either our own team or by trusted colleagues. Moreover, new entries are checked by others before being published.

In its current format (version 1.2) the Computer Virus Catalog contains the following information:

- General information: name, aliases, strain, when and where detected, general classification.
- Precondition: System and models, operating system and versions.
- Easy identification: eg. texts displayed or stored.
- Infection mechanisms, media affected, triggers.
- Modification of the operating system: eg. interrupts hooked.
- Damage: perminent/transient, triggers, special effects.
- Similarities with other viruses.
- Countermeasures as divided into the 6 catagories:
    - Category 1: Monitoring of files, system vectors or areas
    - Category 2: Alteration detection
    - Category 3: Eradication
    - Category 4: Vaccine
    - Category 5: Hardware methods
    - Category 6: Cryptographic methods (hard/software)
- Tested countermeasures and standard means
- Acknowledgements

As the number of antivirus products grew, it quickly became impossible to test all of them. Apart from the catalog entries recieved from others (such as Yuval Tal, or Fridrik Skulason), only our own antiviruses have been mentioned in the catalog. We hope to include the major antivirus packages in the future.

The current version (June-1990) describes 53 MS-DOS, 2 Macintosh, 35 Amiga and 18 Atari viruses.

As German viruses (such as Oropax, Hello, XA1) or other European ones (Vienna, Cascade) form a small (but growing) portion of the international virus scene, VTC Hamburg often gets viruses only some time after the threat appeared elsewhere. We therefore appreciate the help of colleagues all over the world; we are specially aware of the outstanding assistance of David Ferbrache, Christoph Fischer, Yisrael Radai, and Fridrik Skulason, who, by checking new versions of the Virus Catalog, serve as its editorial board.

The catalog is published as a part of Virus Telex, a German Publication similar to Virus Bulletin, and on many e-mail servers throughout the world.

3.4. Limits of these classification methods

On their own, both McAfee's and Radai's classifications are excellent as surveys, but they are not specific enough to help with the identification of a virus. The user will still need full-bodied reports in written form to identify a given virus. They lack information on how to easily detect a virus, exactly what it does, and what to do about it. Recent viruses have also overcome some of the most prominent features, such as file length extension. McAfee's list, the more structured of the two, would have to be expanded (ultimately indefinitely) to incorporate new features of viruses. This would lead to a great many redundant columns.

As mentioned earlier, the Virus Bulletin list offers only little information. The hexadecimal patterns that it publishes can be a valuable help in detecting viruses, but may mislead the user in the case of a virus having the same signature string, but otherwise different.

The Computer Virus Catalog's advantage is the greater detail it offers. It is fairly free-form in structure, which has allowed new types of viruses to be easily incorporated without actually changing the format of the catalog. In many cases, however, the formulation of the entries have left too much room for false interpretation.

4. New efforts

Our new efforts of classifying viruses go in the direction of more advanced uses of the information available on viruses.

4.1. Machine readable version of the virus catalog

As the number of known viruses and virus strains grow, it has become more and more difficult to stay informed on all of the characteristics of viruses - even for us! A solution is to create an information retrieval system that contains information on all viruses. A prerequisate to this is a machine readable form of the present Virus Catalog.

The scope of the machine readable virus catalog entries is roughly the same as in its print form. It is divided into four sections: **index, classification, damage, countermeasures.**

Index:

It includes the same index information such as the name(s) and strain, and platform. An index number has been added as the names given to the viruses are not always unique. This allows the replacing of entry if it need be. This section also includes the information formerly supplied in the acknowledgement section.

eg for the 1701 virus:

```
VIRUS
{
        ID 1704.020.1;                    ;is a unique number given to each virus
        NAME "1701", "Cascade-B", "Herbst", "Autumn"
        FAMILY "1701"
        DETECTED "Vienna", 1988
        CLASSIFIED "Virus Test Center, M. Reinschmiedt"
}
```

Classification:

The next section is the classification of the virus mechansism. We were forced to depart from our previous form of classifying system viruses. System viruses now only include viruses that infect the system without the file system being loaded. On PC·s, this includes boot sector and partition sector viruses, but excludes viruses that only infect system files (such as the Lehigh virus). These are now variations of program viruses.

The next departure resulted from an observation we had made: many viruses have more than one infection target and mechanism. Viruses have been found that target files as well as the boot sector. Many viruses also target the memory for infection, before infecting the files from memory. This is what we previously called 'indirect action'. This form of classification allows for hitherto unknown forms to be classified.

We then go on to describe the effects of infecting each possible target, ie. how much memory is reserved when infecting memory, what interrupts are hooked in the process, when files are infected (on what interrupt), by how much the files are extended, etc.

Such a mechanism record looks like this:

```
(initiator): IF (condition) INFECT (target)
{
      attributes of the infection mechanism and the target
}
```

eg for the 1701-virus:

```
(FILE ON "*.COM"):
        IF ( ) INFECT
        (MCB MEMORY)                ;uses memory control blocks
{
        TARGET LENGTH 1728          ;length in memory
        HOOK INT 21h FN 4B00h       ;hooks interrupt 21h function 4B00h
}

(MEMORY ON INT 21h FN 4B00h):       ;virus is activated in memory by
                                    ;interrupt 21h function 4B00h
        IF ( ) INFECT
        ("*.COM"  AND program.length < 65806)
{
        TARGET LENGTH 1701          ;length in file
        POSTFIX                     ;appends itself to file
                                    ;the virus replaces the first
                                    ;first 3 bytes with a jump statement
                                    ;to the virus
        COPY program[offset 0..2] TO virus[offset 223h]
        COPY &(EBh) TO program[offset 0]
        COPY &(program.length-1701) TO program[offset 1]
}
```

Even if this method of describing the behaviour of viruses seems more complex, it makes an accurate description easier, which is our aim.

Damage:

The damage section is less formal. Here we only describe roughly what is affected. There are text fields where the damage may be further specified. Perhaps this will be expanded at a later date.

Countermeasures:

The last section describes how to identify and remove the virus. This will be described using virus-specific pseudo-code.

Most of this was still under construction at the time of writing, but the structure is not likely to change much.

4.2. A "Threat Description Language: TDL"

The machine readable virus catalog is a mostly descriptive language. At the same time, we are working on a version of the language that contains more algorithmical details: **TDL/V** (Threat Description Language for Viruses). The goal of this project is to create the description of the virus knowledge to be used in an expert system capable of detecting known viruses as well as most unknown variants of known virus strains using data generated by an audit program. The TDL/V entries will be used to generate the knowledge base.

Why an expert system? It has been proven mathematically that there is no algorithmical method for universally detecting viruses. The "viral property" is undecidable and therefore a Universal Virus Detector (UVD) cannot exist, even if many attempts are still made to disprove this very fundamental insight. As in many such problems, the only way to attempt a solution is to try the heuristical approach. If an expert system contains the functional patterns of all (or most) known viruses, an unknown virus that uses a similar pattern will be identified.

5. Conclusion

On September 4th 1989 at its international assembly, the IFIP issued the following recommendations with respect to computer viruses:

> "That in view of the potentially serious and even fatal consequences of the introduction of 'virus' programs into computer systems, the Technical and General Assemblies of IFIP urge;
> all computer professionals to recognize the disastrous potencial of computer viruses;
> all computer educators to impress upon their students the dangers of virus programs;
> all publishers to refrain from publication of the details of actual virus programs;
> all computer professionals worldwide not to knowingly distribute virus code, except in controlled and laboratory environment and all developers of virus detection and prevention systems to stop distribution of virus code for test purposes;
> governments, universities and computer system manufacturors to devote more resources to research into and the developement of new technologies for the protection of computer systems, and
> government to take action to make distribution of viruses a criminal offence." (sic) [IFIP89]

In view of these strong words, things must be rapidly done to help counter ever increasing number of viruses. Existing antiviral protection schemes will eventually fail, if they haven't done so already, in light of recently discovered viruses. To combat the problem in the future, more intelligent tools will be necassary. Although a universal virus detector is impossible, intelligent detectors with up-to-date expert knowledge might be a solution.

One of the dangers the TDL/V may lie within the concept itself: if you have a function language that can describe a virus accurately enough to identify one, wouldn't that enable another expert program to write a functionally identical virus using that knowledge? Or even worse: using the entire knowledge to write a hitherto unknown virus? Such things must be considered and, if possible, protected against.

6. Literature

[Brunnstein 87]	Klaus Brunnstein: "The Informatic Bestiary: Viruses, Worms and other Animals" (in German), Angewandte Informatik, Oct. 1987
[Brunnstein 89a]	Klaus Brunnstein: "Computer Viren Report" (in German), WRS-Verlag, Freiburg, 1989
[Brunnstein 89b]	Klaus Brunnstein: "Zur Klassifikation von Computer-Viren: Der 'Computer Virus Katalog'", Proceedings of the 19th GI (German computer science association) Annual Conference
[Cohen 86]	Fred B. Cohen: "Computer Viruses", Dissertation (PhD), University of Southern California
[Hoffmann 90]	Patricia M. Hoffmann, "Virus Information Summary List", Santa Clara, April 1990
[IFIP 89]	IFIP: "Worldwide Warning on Computer Viruses", San Fransisco, Sept. 1989.
[McAfee 90]	John McAfee: "Virus Chacteristics List Version 57" as published in VIRUS-L and with his product SCANV57.
[Radai 89a]	Yisrael Radai: "PC/MS-DOS Viruses, May '89"
[Radai 89b]	Yisrael Radai: "PC/MS-DOS Viruses, May '89"

Appendix 1:

------- Computer Virus Catalog 1.2: "Virusname" (Date of Entry) -------	
Entry	: "Virusname" (=Name of virus)
Alias(es)	: Alternate Name(s)
Virus Strain	: "Family" (if any) to which this virus belongs
Virus detected when	: Date of first appearance
where	: Where was Virus produced or first detected(both entries only if well-known)
Classification	: System Virus (BootSector, Command.Com, BAT V.) Link or Program Virus (Overwriting/Extending V.) Resident, Direct Action
Length of Virus	: 1.Length (Byte) on storage medium 2.Length (Byte) in RAM
------- Preconditions -------	
Operating System(s)	: e.g. AMIGA-DOS, ATARI-TOS, MacOS, MS-DOS, UNIX, VMS, MVS, VM
Version/Release	: Special Version of OS (e.g. UNIX System V, UNIX BSD, VMS etc) if needed, and Release (e.g. MS-DOS 3.2, UNIX BSD 4.2)
Computer model(s)	: The Computer models (e.g. ROM BIOS versions) on which the Virus runs.
------- Attributes -------	
Easy Identification	: if applicable: Typical texts, either messages (e.g. screen), or texts in Virus body (readable with HexDump-facilities), Volume Labels etc. by which viruses may easily identified
Type of infection	: Self-Identification methods; Executable File infection(.COM,.EXE):overwriting, extending; resident; (RAM/File) Direct Action; WCS infection (e.g. CMOS at initialisation setup); System infection: RAM-Resident, Reset-Resident, Bootblock/Bootsectors, Command.Com, BAT, Device Handlers/Libraries etc; Infection of unlinked Object Files; Source Code Infection.
Infection Trigger	: e.g. time/date, other events, random, reset (CTRL+ALT+DEL), operations such as: DIR, execution of specific program (.COM/.EXE). Storage media affected: Infection of (particular) diskettes, hard disks, DiskPacks, etc.
Interrupts hooked	: Interrupts used and changed by this virus.
Damage	: Permanent Damage: e.g. overwriting bootblock, repeated restart/format, zeroing of sectors, Bad Sectors in FAT etc; Transient Damage: e.g. screen buffer manipulation, audio effects, blinking LEDs; Transient/Permanent Damage: viruses which (under specified conditions) produce parmanent damage while "normally" producing transient damage.
Damage Trigger	: e.g. time/date, value of infection counter, other events, random, reset, operations.
Particularities	: special effects e.g. process velocity slowed-down
Similarities	: dis/similarities to other viruses (either from same "family" (=strain) or different viruses); names of related viruses.
------- Agents -------	
Countermeasures	: Names of tested products of Category 1-5: Category 1: 1 Monitoring Files: program which monitors (attempted) changes in files 2 Monitoring System Vectors: program which monitors changes in vectors (e.g. resident, Interrupt vectors) 3 Monitoring System Areas: program which monitors System Areas such as BootSectors/Blocks. Category 2: Alteration Detection: a program which detects changes in given files Category 3: Eradication: a program which erases a specific virus code from files or from RAM (if resident) Category 4: Vaccine: a program which alters files (on permanent storage) or RAM resident programs such that viruses regard them as already infected Category 5: Hardware Methods: methods to detect or prevent alteration or infection of files, vectors or system areas. Category 6: Cryptographic Methods (Hard/Software): methods keeping programs on storage in encrypted form, and decrypting them before execution.
-ditto- successful	: Names of those countermeasures (of given category) which, without (or with known "small") restrictions or side effects, were "successful" to detect, identify, inactivate or erase the given virus or exclude infection by it.
Standard means	: Means in the respective System which may be used to identify/destroy this virus.
------- Acknowledgement -------	
Location	: e.g. Virus Test Center, University Hamburg, FRG
Classification by	: Author(s) of Reverse-Engineering Document
Documentation by	: Author(s) of this Catalog Entry; Translator of Non-English document (if applicable)
Date	: Production/last Update of this Catalog Entry (this information also in the 1st line)
Information Source	: Information used for Documentation (only in cases where Reverse-Analysis was not possible).
------- End of "Virusname"-Virus -------	